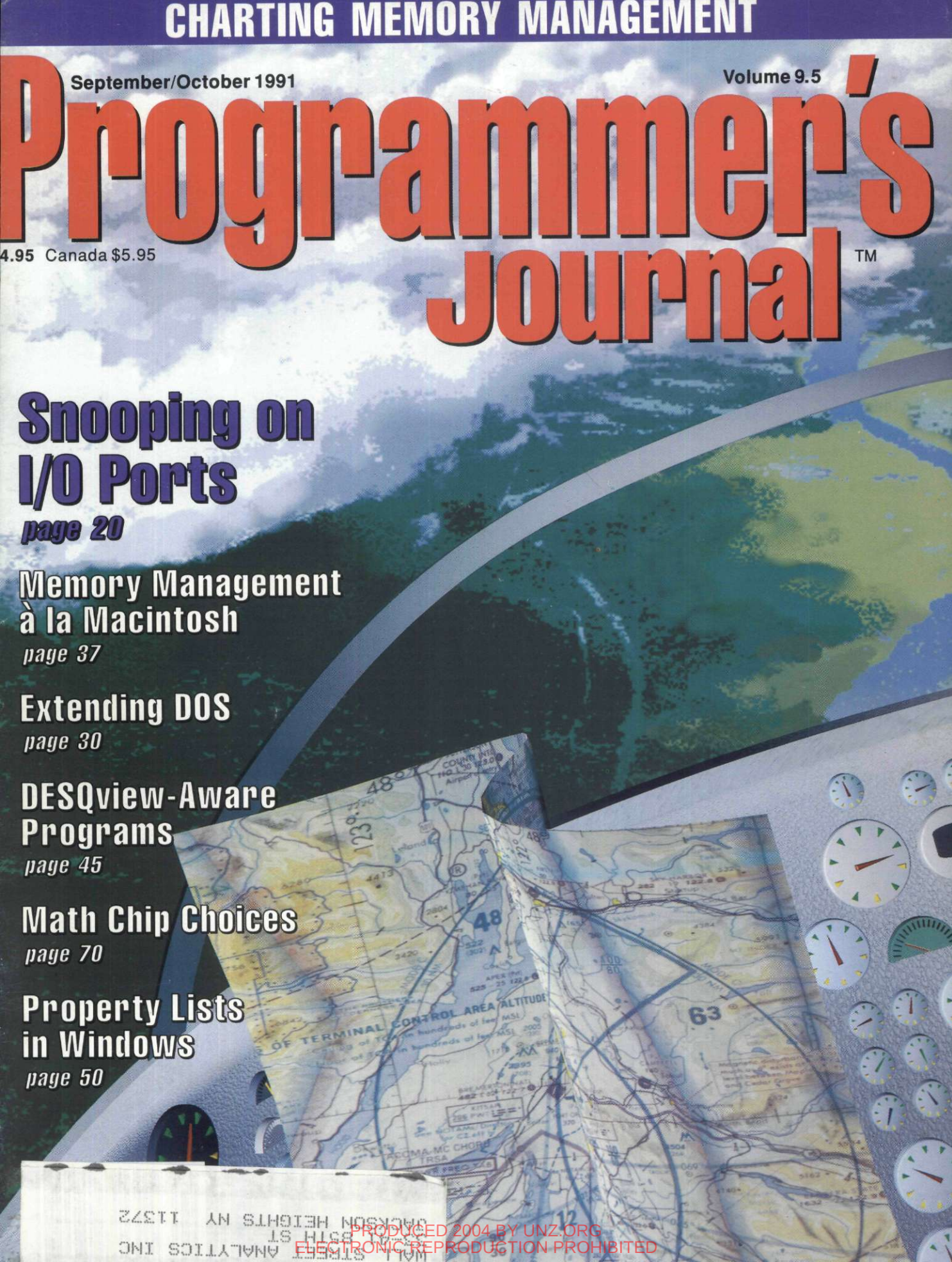
CHARTING MEMORY MANAGEMENT

September/October 1991 Volume 9.5

Programmer's Journal

4.95 Canada $5.95

TM



Snooping on I/O Ports
page 20

Memory Management à la Macintosh
page 37

Extending DOS
page 30

DESQview-Aware Programs
page 45

Math Chip Choices
page 70

Property Lists in Windows
page 50

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Recycle

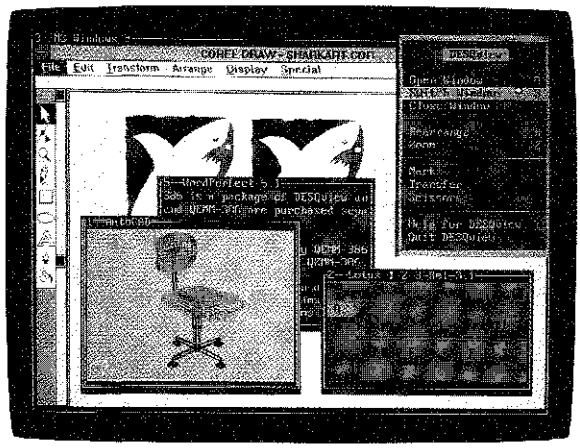
Everyone wants more productivity out of their computers.

But not everybody is willing to spend a fortune to get it. That's why Quarterdeck productivity software is outselling everyone else's—including Microsoft's.

DESQview Multitasks and Windows on Your PC

As early as 1982, computer enthusiasts found our products helped them set up their ideal working environment. Since then, step-by-step, we've improved DESQview into what some very knowledgeable people call "the best alternative to OS/2".

Today's DESQview allows you to multitask multiple programs in windows side by side—text programs; graphic programs. But best of all, you don't have to buy a lot of new hardware and software to use it. DESQview works with the PC and the programs you now own.



Over 1,000,000 users multitask their programs with DESQview.

QEMM Breaks the 640K Barrier

Break the 640K barrier in DOS—or in Windows 3.0—and give your DOS programs up to 130K more room within the first megabyte of memory, plus another 96K of video memory, in some instances.

QEMM takes buffers, network drivers, TSRs and other memory-using utilities and moves them into idle areas between 640K and one megabyte.

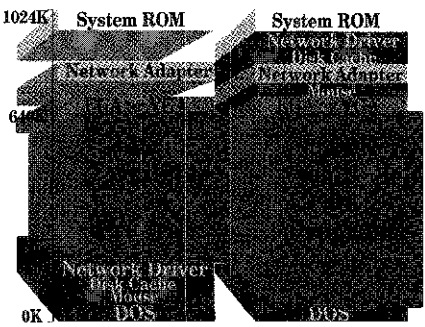
It's not complicated. It's easy. Just type *Optimize* and QEMM does the rest.

Number one.

QEMM 386 is included with DESQview 386 to create the optimum operating environment for productivity oriented users of 386, 386SX and i486 PCs.

QEMM 50/60 is designed to work in IBM® PS/2™ Model 50 and 60 PCs with specific IBM adapter boards.

QEMM is the #1 selling utility according to distribution sources. In fact, it was the number one selling software package in the PC industry in April, May and June 1990.



Before *After*

INFO WORLD Product of the Year 1986, 1987, 1988, 1989

PC WORLD BEST BUY

PC MAGAZINE EDITORS' CHOICE

PC MAGAZINE EDITORS' CHOICE

PC Computing MVP MOST VALUABLE PRODUCT 1990

PC TECH Professional Solutions Award 1989 Best Operating Environment

1989 BYTE AWARDS OF EXCELLENCE

These are some of DESQview's most recent awards

QEMM 386 PC WEEK PRODUCT EVALUATION SCORE 8.6

INFO WORLD BEST IBM ITS CLASS Dec. 18, 1989

INFO WORLD TOP 100 1989

1989 BYTE AWARDS OF EXCELLENCE

PC Computing MVP MOST VALUABLE PRODUCT 1990

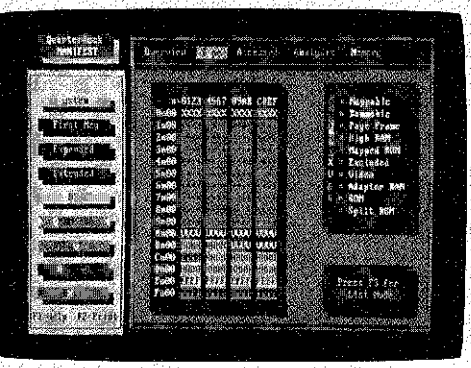
These are some of QEMM's most recent awards

©1991 Quarterdeck Office Systems. IBM and PS/2 are trademarks of IBM Corporation. PC Tools Deluxe is a trademark of Central Point Software. 80286, 386, 386SX and i486 are trademarks of Intel Corporation.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

your PC

Manifest Gives You In-Depth Knowledge of Your PC



Our newest utility is Quarterdeck Manifest, the best way to discover everything you ever wanted to know about your PC. Manifest shows you around 'under the hood', pointing out how memory is used, comparing memory speeds, and indicating how you can gain more room for your programs to work.

It can point you to as much as 130K of additional RAM your programs can use. It shows you which memory areas are faster. It even helps you compare add-in memory board performance.

Manifest does for memory what PC Tools Deluxe does for disks. And it's easy to use.

Manifest has been quick to win recognition

QRAM Optimizes the Memory of 8088 and 80286 PCs

Once you know where you can move those memory-hogging utilities, QRAM lets you do it. It even works on 8088, 8086 and 80286 PCs with EMS 4.0 or EEMS memory boards.

QRAM and Manifest help you get every last 'K' out of the hardware you own.

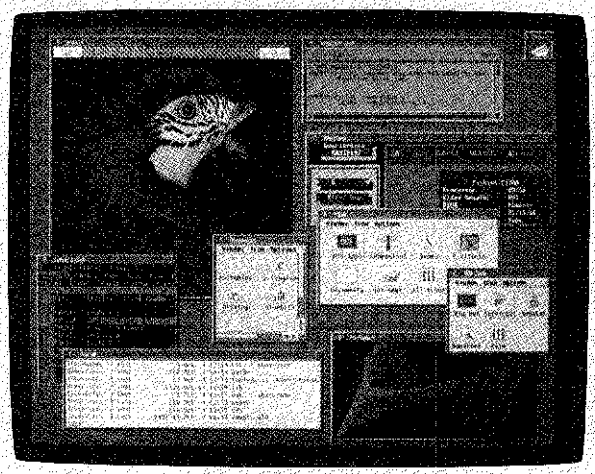
In fact, all our products are designed to give you more productivity out of the system you already have, whether 8088, 8086, 80286, 386SX, 386, or i486.

Quarterdeck Products for Everyday Heroes

Our mission is to protect your investment. Whether your PC is 9 years old or fresh out of the box, our products make it more versatile, more flexible; and help deliver performance dividends from your computer investment. Our products help fine-tune your PC which helps you do your work better, which makes you look good.

A Glimpse of the Future: DESQview/X

We're also looking ahead to the next wave of computer system development: enterprise computing. Our new



DESQview/X allows different computers with different operating systems to work together. Using the advanced X-windows environment, it lets users run programs on remote computers and watch them run in their PCs' windows. DESQview/X will be available later this year.

Quarterdeck products. The best way to get the most out of your PC today. And tomorrow.

Quarterdeck

150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax (213) 399-3802

Circle 1 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Do You Love COBOL?

Then the COBOL Report Activator *(COBRA)* System is for you!

The ***COBRA*** System generates textbook quality, structured COBOL programs in seconds. No more reasons for 'quick and dirty' programming - use ***COBRA*** for 'Quick and Clean COBOL'®.

With the ***COBRA*** System you can:

- ☆ Use your existing copybook members
- ☆ Use sequential, indexed and BTRIEVE files
- ☆ Use one or two files for input
- ☆ Add COBOL Code in seven places before you generate a COBOL program
- ☆ Modify the generated COBOL program
- ☆ Tailor the System defaults to fit your site

The ***COBRA*** System will generate code for internal sorts, audit trail reports, and up to five output files. Record selection logic is easy - include or exclude records as desired.

The **new** Version 4.0 of the ***COBRA*** System is available in both PC and mainframe versions.

Do You Hate COBOL?

Then the COBOL Report Activator *(COBRA)* System is for you!

The ***COBRA*** System is a keyword driven, free-form method to generate complete COBOL programs! Use the ***COBRA*** System and any COBOL compiler to write reports or reformat files. Just a few short SQL statements and the program is written for you.

Using the ***COBRA*** System, you have:

- ✓ No Divisions to worry about!
- ✓ No logic to figure out!
- ✓ No syntax/compile errors to fix!
- ✓ No logic errors to debug!
- ✓ No hassle!

The FD Conversion Utility (included) will convert existing COBOL File Descriptions into the ***COBRA*** System data dictionary format automatically.

The ***COBRA*** System is easy to learn and use. With the examples in the manual and online HELP file, you can generate your first program in less than one hour from opening the package.

Here are the only statements required to generate a complete COBOL program - an internally sorted, control-break totals report:

```
TITLE SAMPLE REPORT FOR MULTIPLE CONTROL BREAK PROGRAM
REPORT TOTALS
SELECT EMPLOYEE-NBR, EMPLOYEE-LAST-NAME, EMPLOYEE-FIRSTNAME,
       ADDRESS,      THIS-PER-EARNINGS,  YEAR-TO-DATE-EARNINGS
FROM EARNINGS-FILE
ORDER BY PLANT-CODE, DEPT-NBR, EMPLOYEE-NBR
GROUP BY PLANT-CODE, DEPT-NBR
```

☞ Special Offer for PJ Readers!

Regular prices for the ***COBRA*** System are listed on this page. As a PJ reader, you will receive a 20% discount on all systems. You also receive our No-risk, 30-day money back guarantee. You must mention this ad when ordering. **Offer expires October 31, 1991.**

To order, call 1-800-COBOL-4-U

COBRA System Pricing:

IBM PC (DOS)	$ 495.00
PC-Site License*	4,950.00
IBM DOS/VSE	14,950.00
IBM MVS/XA	19,950.00

Other system prices upon request

*up to 20 users

"An incredible time saver. I like it!" - Tim McCormley, **acucobol** technical support

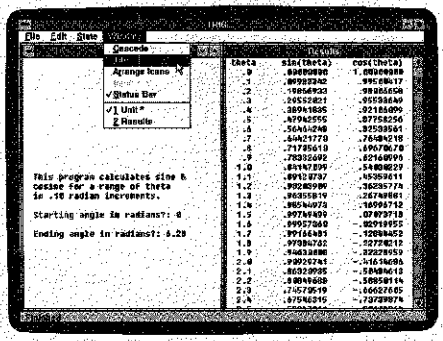
Olivadoti Associates, Inc. PO Box 750817 Petaluma, CA 94975 **(707) 763-5123**

Circle 3 on Reader Service Card

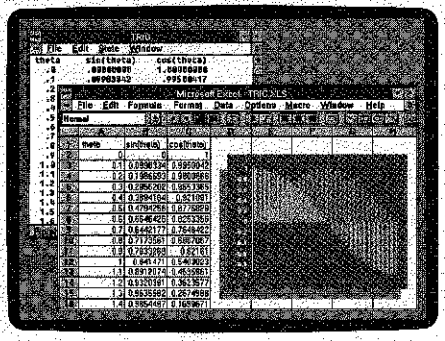
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MICROSOFT SOFTWARE DEVELOPMENT TOOLS

Windows helps FORTRAN users create bigger apps.



The Microsoft Windows graphical environment lets you have multiple windows in your FORTRAN applications. This means that you can resize and scroll input and output displays to view data...



...and to analyze data, simply cut and paste your FORTRAN output into programs such as Microsoft Excel for Windows and you can view it instantly in graphical form.

With the new Microsoft® FORTRAN Professional Development System version 5.1, your existing code taps into the power of the Microsoft Windows™ graphical environment. FORTRAN 5.1's new QuickWin library lets you develop 16-bit apps that access greater memory than ever before – breaking the 640K barrier on 286 and 386 machines.

You can use the FORTRAN QuickWin library to take advantage of multiple I/O windows, multitasking, and cut and paste. Or write FORTRAN routines called from mixed-language Windows apps. And FORTRAN 5.1 has Programmer's WorkBench, with a new CodeView® debugger and Source Browser.

To see the advantages of bigger, multi-windowed 16-bit applications, give us a call at (800) 541-1261, Department R38 and order your FORTRAN update today.

"As used herein, "DOS" refers to MS-DOS or PC-DOS operating systems. © 1991 Microsoft Corporation. All rights reserved. Printed in the U.S.A. Inside the 50 United States, call (800) 541-1261, Dept. R38; outside the 50 United States, call (206) 936-8661. Customers in Canada, call (416) 568-3503. Microsoft, MS-DOS, OS/2, CodeView and the Microsoft logo are registered trademarks and Windows and Making it all make sense *are trademarks of Microsoft Corporation.*

Key Features

- Create Windows .DLLs in FORTRAN using new or existing code.
- QuickWin Features:
  - QuickWin child windows are easily created using the OPEN statement.
  - User-defined positioning and titles for child windows.
  - Automatically generated scroll bars for output that extends past a single screen.
- CodeView debugger supports DOS*, Windows-based and OS/2® applications.
- Extended CodeView debugger for large DOS programs.
- Complete online documentation for the FORTRAN language and all compile and link switches.
- DOS and OS/2 run-time libraries are compatible with other Microsoft languages.
- Floating-point support includes co-processor, emulation, and alternate math libraries.
- 100% ANSI 77 compatibility and numerous IBM®, VAX® and ANSI 8X extensions.
- New BYTE keyword emulates VAX data types.
- Language Extensions include DOUBLE COMPLEX variables, precision and maxexponent inquiries.

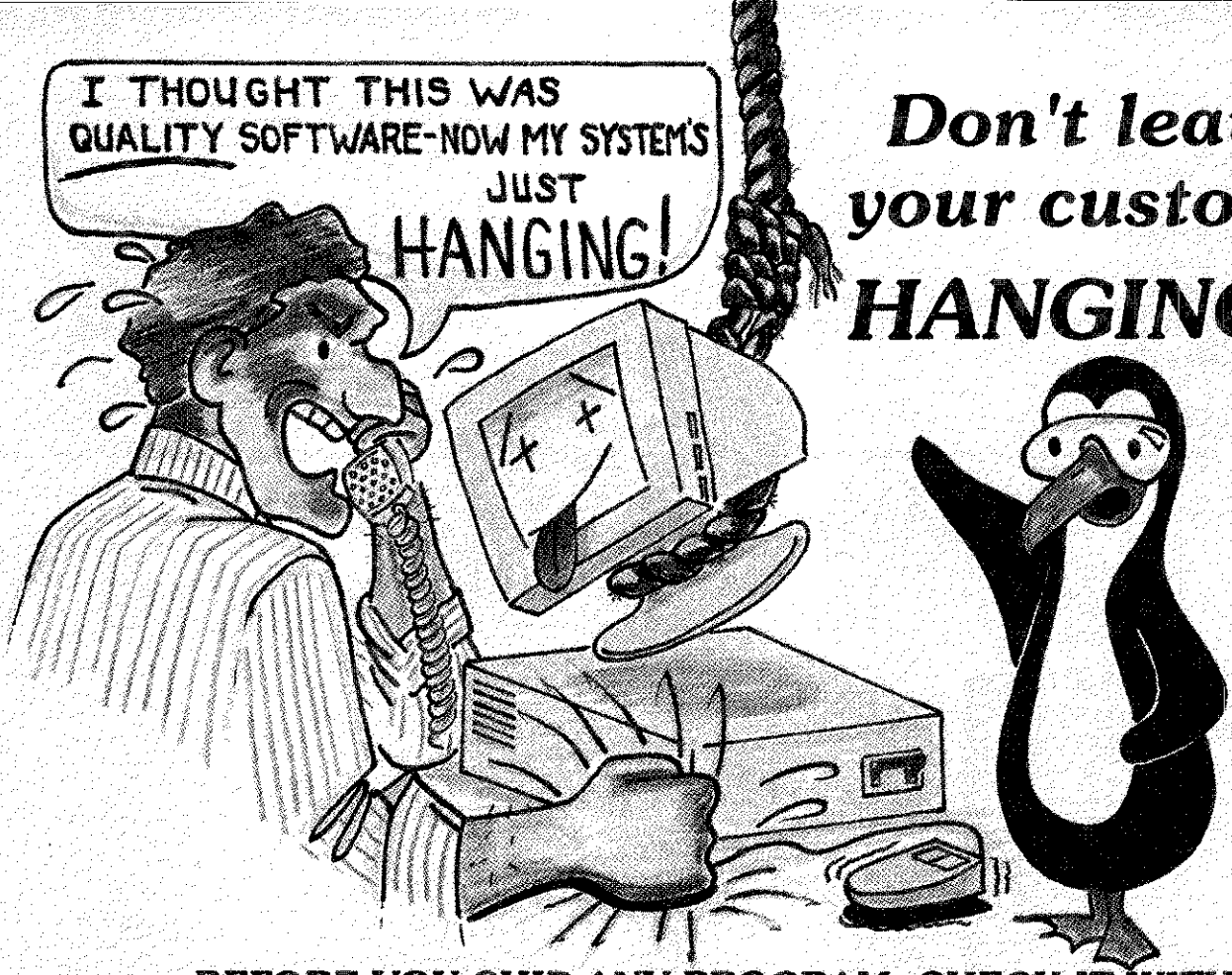
PROGRAMMER'S TIPS

- *Use the new /MW option with the FL command to invoke the QuickWin library. For example: "FL/MW MYAPP.FOR" is all it takes to make MYAPP a Windows-based program.*
- *Use the ALLOCATE statement to dynamically size arrays and to access more than 16MB of memory on a 386.™*

Microsoft
Making it all make sense®

Contact this advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Don't leave your customers HANGING. . .

BEFORE YOU SHIP ANY PROGRAM, CHECK IT WITH

COMPUTER LANGUAGE PRODUCT EXCELLENCE AWARD 1990 COLA

BOUNDS-CHECKER™

WINNER PC MAGAZINE BEST OF 1990

JANUARY 15, 1991 PAGE 168

FINDS OUT-OF-BOUNDS MEMORY ACCESSES *AUTOMATICALLY*

Your program may have 10,000 to a million lines of code. It may occasionally hang mysteriously or it may appear to run flawlessly every time. But under DOS, how can you ever be sure that your program is not corrupting memory it does not own? The only way to be 100% sure is to BOUNDS-CHECK before you ship.

To use BOUNDS-CHECKER you build your program with debugging information (we support most compilers including Microsoft, Borland & JPI). Then you just type <BC file-name>. BOUNDS-CHECKER sets up the 386™/i486 for protection and lets your program fly. If your program accesses memory it does not own or overwrites its own code, BOUNDS-CHECKER pops up displaying the offending SOURCE-LINE or instruction.

Programming under DOS is a gamble, so why not stack the odds in your favor--CALL TODAY.

(603) 888-2386

OTHER FINE NU-MEGA PRODUCTS. . .

The ultimate systems debugger.

Debug:
- Interrupt routines
- Device drivers
- T&SRs
- ROMS
- Applications
- Overlays

Soft-ICE™

Features:
- Break out of a hung program
- Real time Break-Points
- Back-Trace history
- Works with other debuggers

COMING SOON
Soft-ICE for Windows

Get inside Windows with the first REAL Windows system level debugger. $386

1988 BYTE AWARD OF DISTINCTION

Run CODEVIEW for Windows on a single monitor
CV/1.....$129
• Runs in a window • No annoying flash •
• Runs on any display that supports windows •

Call by 4:00 PM EST TODAY and ask us to EXPRESS you an info packet. In most cases you will receive it by 10:30 AM tomorrow. (USA only)

All Nu-Mega products require a 386, 386SX or 486. MS-DOS and Codeview are trademarks of Microsoft Corp. 386 is a registered trademark of Intel Corp. Nu-Mega, BOUNDS-CHECKER, Soft-ICE and CV/1 are trademarks of Nu-Mega Technologies, Inc.

Nu-Mega™ TECHNOLOGIES INC

P.O. Box 7780 • Nashua, NH • 03060-7780 U.S.A.
(603) 888-2386 • Fax (603) 888-2465

BOUNDS-CHECKER $249
Soft-ICE $386

Special Offer
Buy BC & S-ICE .....Save $100

30 Day Money Back Guarantee

Circle 4 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

September/October 1991 Volume 9.5

You don't have to fly blind through the ever-expanding terrain of operating systems. Use this issue to chart a smooth flight path to optimal memory management.

Happy landings!

Cover Art: Digital Artworks

FEATURES

SYSTEMS **20** **EAVESDROPPING ON I/O** *By Rick Knoblaugh*
Your mother taught you it wasn't nice to eavesdrop, but with Rick's code and the virtual-8086 mode of the 386, you won't be able to resist checking out what your I/O ports are up to.

SYSTEMS **30** **DOS EXTENDERS** *By M. Steven Baker*
DOS extenders continue to open up new horizons for the 286 and 386. The hot new phrase is DPMI compatibility, which allows extended DOS apps to run under Windows 3. Steve weaves through the maze of new products available for stretching the limits of DOS.

SYSTEMS **37** **MANAGING MEMORY** *By Brett Glass*
The early Macintosh computers were born in an age of limited memory, so Apple's system designers incorporated a sophisticated memory-management system. Brett looks at the strengths and limitations of this Mac-style, handle-based memory scheme.

SYSTEMS **45** **DESQVIEW-AWARE PROGRAMS** *By Murray L. Lesser*
Don't turn off Murray's tailor-made ALARM program—it might help keep track of your schedule and free you from watching the clock while under the spell of the computer muse. Along the way, learn how to take advantage of DESQview's multitasking capabilities by mixing assembly language with BASIC.

WINDOWS **50** **WINDOWS PROPERTY LISTS** *By William S. Hall*
DOS isn't the only operating system that fences you in. Bill shows how to overcome Windows memory-management problems with the right properties in hand.

GRAPHICS **61** **ICON POWER** *By Peder Jungck*
Walk like an encryption and look like a wizard—Peder shows how to use VGA write mode 1 to draw icons with lightning speed. Avoid the bus bottleneck—take a magic carpet ride instead.

NUMERICS **70** **DIVIDE AND CONQUER** *By Mark J. Barrenechea*
Mark evaluates floating-point coprocessor options for the 80386 and 80486, comparing the performance of I/O and memory-mapped math chips. Although the trend in the nineties is toward tightly integrating the floating-point and integer execution units, his test results provide some surprises.

C **76** **WINDOW LAYERS** *By Allen Holub*
In this installment of his windowing system, Allen makes it clear that every window has its layers. Learn to see your way through the viewport and text buffer to the underlying file.

COLUMNS

CONSULTING **10** **TQM: THE NEW BELTWAY BUZZWORD** *By Paul Barkley*

PJ FORUM **17** **CHIPS OFF THE OLD BLOCK** *By Dave Thompson*

ARMCHAIR **87** **PURSUING SYSTEMS PROGRAMMING** *By M. Steven Baker*

CONSULTING **89** **THINKING ABOUT PROJECT MANAGEMENT** *By William H. Roetzheim*

LANGUAGES **91** **A SOFTWARE FABLE** *By Richard Riehle*

OFFRAMP **96** **PLAYING FOR KEEPS** *By Hal W. Hardenbergh*

DEPARTMENTS

6 **EDITOR'S NOTE**

8 **LETTERS**

48 **BBS LISTINGS**

49 **PJ CONTEST**

80 **AD INDEX**

92 **PRESSROOM**

94 **PJ BILLBOARD**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

Well pardners, it's time again for my written ramblings. You may have guessed that my previous training had no connection with publishing. I did write a musical, "Florida Cowboys," in fifth grade, but the theater circuit never picked it up. Then I was going to be a "creative scientist." An architect and engineer by training, I stumbled into the computer business. A few years back I started editing and writing, and I've been observing the industry from this vantage point ever since.

The time seems ripe for exploring systems programming on the PC. Microsoft has released MS-DOS 5, a long-overdue improvement over its half-broken predecessors. One assumes it wasn't by chance that Microsoft's DOS 5 arrived before DR-DOS 5 from Digital Research (now owned by Novell) could catch the wave.

The split between Microsoft and IBM is getting lots of attention. Any day now we'll have Windows 3.1 bucking out of the bronco pen, sporting pen-based computing, multimedia extensions, and TrueType fonts. Meanwhile, back at the Boca Raton ranch, IBM has promised OS/2 2.0 by year's end. Their OS/2 Developer's Assistance Program includes free beta product to both small and large developers, OS/2 porting technical support, and hardware loans (PS/2s, we presume).

This new OS/2 version looks promising. It has a DOS compatibility box that works, and support for DPMI and the Windows 3 API is forthcoming. Unfortunately, many developers got burned believing Big Blue's previous promises for OS 1.2. We'll soon know whether IBM can deliver the goods and go head-to-head against Microsoft's DOS and Windows. The skirmish should last until Windows 4 (WIN32) and DOS 6 are released. Because version 2 will support the Windows 3 API and DPMI, developers can program for Windows or a DOS extender and still run under the new IBM OS without ever writing OS/2 or Presentation Manager code. This means that DOS and Windows remain safer bets for many developers.

NEW DOS EXTENDERS RIDE INTO TOWN

Inertia is a powerful force, which accounts for the incredible staying power of DOS. Before we move to the 586, software developers should take advantage of the 386. So DOS extenders are turning up in more and more products. Most DOS extenders will soon be updated to support DPMI. Phar Lap was first out of the gate with new 286 and 386 releases, soon to be followed by Rational and Ergo. These new versions allow you to build applications that run under Windows as DOS apps. Several 386 compiler vendors (Watcom, MetaWare, and MicroWay) will be offering products to support a 32-bit Windows API from the 386 protected mode (no small feat).

Intel's C Code Builder is bundled with a DPMI-compatible version of the IGC DOS extender. I'd keep an eye on Silicon Valley Software (SVS); they're including the same Intel (IGC) DOS extender with their C, Fortran, and Pascal language products, and their prices are very attractive. Both offer free runtimes as well.

System 7 will keep Apple developers and users busy for a while. The 386 Unix market is now shipping System V Release 4.0, which merges the BSD variant with the AT&T strain. Some vendors toss in X Window and either the Open Look or OSF Motif GUI interfaces. Fortunately, the hardware is keeping ahead of the gluttonous software: if you want Unix 4, expect 80 disks or a 120MB tape cartridge. Sun discounts CD-ROM readers so that they can ship all their OSes on a single CD.

THE CORPORATE GIANTS AND THE UPSTARTS

The hottest news these days is that the big guys are cutting deals with each other like crazy: Apple and IBM set up a partnership to develop the OS of the future, IBM and Wang sell mutual products, Borland buys Ashton-Tate to become number three in the U.S. software business (just behind Lotus), Novell buys Digital Research. The prevailing assumption is that there is safety in numbers (of dollars and shares)—the bigger they get, the less often they fall.

But as the market expands, new crevices form in which small companies can get a foothold. The Windows developer tool market is a prime example of this kind of opportunity. Who knows what fertile ground these OSes will provide, or what new products will rise up from them?

Steven Baker

Programmer's Journal

Editor
M. Steven Baker

Managing Editor
Cecelia Hagen

Associate Editor
M. Elizabeth Hardinger

Contributing Editors
Michael Abrash, Brett Glass, Frank D. Greco, John Otken, Tom Swan

Founder
Greg Estes

Publisher
Elizabeth Oakley

Advertising Manager
Jay Moore

Account Executive
Maryann Kearns

Advertising Coordinator
Bonnie Chin

Circulation Manager
Randy Randolph

Production & Art Director
Wanda Walker

Production Assistants
Margie Hanna, Nicole Harris

Financial Controller: Thomas Graham
Subscriptions: Pamela Coleman
Office Staff: Tina Bates, Barbara Curtright, Jill Edwards, Bruce Searl

Business and Editorial Offices: ***Programmer's Journal,*** P.O. Box 70167, Eugene, OR 97401-0110, (503) 747-0800, (800) 234-0386, Fax: (503) 746-0071, BBS: see page 48

Subscriptions: 1 year (6 issues) $29.70

Change of Address: ***Programmer's Journal,*** P.O. Box 70387, Eugene, OR 97401-0121

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P. O. Box 70167, Eugene,OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher, Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000 293. **POSTMASTER:** Send address changes to *Programmer's Journal*, P.O. Box 70387, Eugene, OR 97401-0121.

© 1991 *OAKLEY PUBLISHING COMPANY*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MetaWare

Multi-Language, Multi-Platform Compiler Products for Professional Software Developers

2161 Delaware Avenue • Santa Cruz, CA • 95060-5706 • (408) 429-6963 • FAX (408) 429-9273

Copyright © 2004 by UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

WE ALL LAUGHED

Is your correspondent Mr. Stanislav a humorist, or what? In his letter (*PJ* 9.4) he states that he writes about 2000 lines of code a day. If each line is 20 characters, he types about 40,000 characters a day, or about 5000 characters per hour. This equals approximately 16 words per minute, which is about one-third the speed of an average typist entering directly from copy.

Of course, that's probably an easy program of 40,000 or 50,000 lines, such as a complete general ledger program. For more difficult stuff, like a multitasking operating system, Mr. Stanislav would probably slow down. With such programming speed available, it's surprising that the rest of us low-productivity programmers aren't out on the streets.

Could Mr. Stanislav be enticed to write an article for *PJ?* Suggested titles: "A StarWars drop-in module" (available on 256,000 CD-ROMs); "KEYPRO: Make your 'A' key write structured modules"; or "The meeting that prevented me from writing OS/2 4." Naturally, I have hidden this issue from my boss—I don't want to be replaced by Mr. Stanislav working during his coffee break.

Seriously, we need humor in our dull, drab programming lives, and I'm glad that *PJ* has made our programming staff fall all over themselves laughing.

JP Harrison
Hood River, OR

VEXED BY HEX

The "Hexadecimal number conversion" sidebar in Mark Barrenechea's article (*PJ* 9.3) explaining the conversion of the hex value 4CEB79A3 to single-precision floating point committed two egregious errors. The result was shown as 493827168.0; the correct result is 123456792.0.

The first error came from the computation of the unbiased exponent: the bias is 127, not 128. Therefore, the unbiased exponent is 26, not 25.

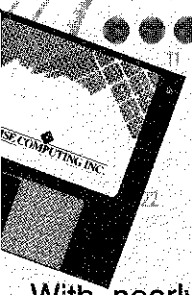
The second error came from the shift of the normalized significand to the finalized binary form. The radix point (not the "significand," as stated) was shifted right 28 places—not 26, or even 25.

The hex representation of the number 493827168.0 is 4DEB79A3, which differs in one nibble from that given in

HAHAHA HA HA HA HA HA HA HA HA
HA HA HA HA HA HA HA HA HA HA
HA HA HA HA
HA HA HA
HA HA HA

WE'VE CHARTED THE WAY WITH WIN++

...for Windows applications in Borland C++



We've chosen the direction and plotted the course to easy and powerful Windows applications development!

With nearly 100 classes, Win++™ provides a simplified, flexible and powerful high-level C++ interface making your Windows development efforts a breeze. Other classes for setting internal data structures and integrating data into applications are also included.

Put the wind at your back...

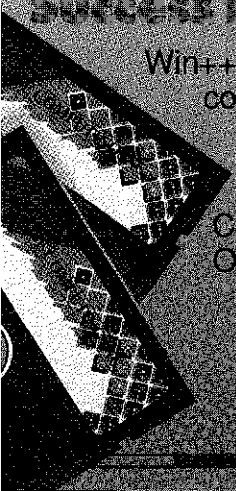
Central to Win++ is that it provides a class for each type of display object in Windows. Win++ includes ready-made classes for representing dialog boxes, child controls, pens, brushes, menus, bitmaps, the cursor, icons, a printer, the clipboard and many others.

Win++ also supports the construction of dynamic-link libraries (DLLs). Use Win++'s ready-made DLLs in your programs and save valuable time, memory and disk space. And, you may distribute Win++'s DLLs with your applications.

It's smooth sailing from here out...

Win++ comes with a comprehensive reference manual and tutorial. Complete source code is included. You can rely on Blaise Computing to provide and support quality add-on libraries. We've been doing so since 1983 — it's our business — our only business.

Success is in the offing...



Win++ requires Windows 3.0 and Borland C++ Professional version 2.0 or later. Win++ costs just $249. We're so convinced that you'll find Win++ essential to your applications that if during the first 60 days you're not completely satisfied, we'll refund your money. That's a 60-day money back guarantee!

Call our order department toll free at (800) 333-8087, today!
Or, by fax, (415) 540-1938.

BLAISE COMPUTING INC

Trademarks are property of their respective holders.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TQM: The new Beltway buzzword

By Paul A. Barkley

When the massive Desert Storm victory parade marched through D.C. in June, I didn't see a TQM float on the Mall, but there might as well have been one. You're going to hear more and more stories from TQM seminar leaders about how the war was won because this or that small supplier ran their businesses using TQM. You may not hear TQM and Patriot missile in the same sentence, but you're no doubt going to hear about TQM and the parts suppliers for those missiles.

TQM stands for Total Quality Management, and the federal government is going after it in a big way. It starts with a few government managers who come back from a TQM seminar with some ideas. At first, they encounter resistance, particularly from the long-term bureaucrats, but pretty soon a few managers here and there have tried some of the ideas and found some success. The higher-up managers get the TQM religion, send their subordinates off to classes, and so it goes. The upshot of all this is that it's no longer enough to submit the lowest bid. Firms that want federal contracts are expected to deliver high-quality goods and services as well. That's a tall order.

Meanwhile, consulting and contracting firms are doing business as usual: finding more and more ways to cut costs and fighting the "o" word ("overhead"). Eventually, it will become obvious that some sort of attention to quality is necessary to keep the clients happy. The *Challenger* disaster was an expensive lesson in the importance of quality versus the lure of cutting corners.

Maybe "business as usual" is about to change. Maybe one day the government will start requiring successful bidders to have a TQM philosophy in place. Until then, if TQM works half as well as people say it does, it will be a real competitive advantage.

ILLUSTRATION BY JIM CARPENTER

WHAT IS TQM?

I don't pretend to be a TQM expert. It has too many aspects to assimilate at one time, and a disturbing amount of the TQM interpretation seems contradictory—sort of like a Kahlil Gibran poem. Gibran would explain what a tree was like by saying that a tree was sort of like the daytime, and then again sort of like the nighttime. By the time he finished, you didn't have the vaguest idea what a tree was like, but you knew you wanted to strangle him. TQM is a lot like that.

TQM is a philosophy of doing business. An American named W. Edwards Deming dreamed up TQM 30 years ago, but the Japanese were the first to see its potential. It's a way of looking at the job, not just a few isolated principles to put in place. If you embrace one of the martial arts, it is said, you'll find a calm center and learn to deal with stress. (Of course, you'll also be able to break bricks with the side of your hand.) Similarly, TQM will have you looking differently at the way you work, searching for new ways to do things that you once took for granted. TQM isn't accomplished any faster than a black belt is; changes in your organization will take time.

CONTINUOUS IMPROVEMENT

To me the most important principle of TQM is continuous improvement. You must never be satisfied with the status quo, because you can always increase the quality of your product or service.

TQM's opposite is the "If it ain't broke, don't fix it" approach. One company I worked for was developing Clipper programs on a large contract. Some of the people had been working on this contract for many years, so there was a substantial contract history and an accompanying attitude of inertia.

I started talking about the need to develop a set of reusable modules with increased usefulness. The work, organized by end-user program offices, was fragmented, so that the only people who could maintain the systems were those in the same small office team. Yet some of the best coders were perfectly content with what they were doing. They were turning out "pretty good" systems. The clients weren't complaining, and the systems functioned decently. They had been developed fairly efficiently because the coders, like all experienced coders, kept improving their modules with each new system, starting as much as possible with the old modules and then adding a little tweak here or there.

What the coders didn't realize was that no one had an *overall* best system. When we began to discuss each module, ideas started to flow. For years, I've written systems without a backup and restore utility built in; I let my users handle this with outside methods. Other coders were nicer to their users: they had simple routines to copy files, assuming that they could fit on a single floppy. One programmer had a routine to figure out how much room there was on the floppy. It would then write out DBF records one at a time until the disk was filled. Records would continue writing on the next disk, thus eliminating the problem of handling multiple

Paul Barkley can be reached at 13312 Neerwinder Place, Germantown, MD 20874.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Craftsmanship Isn't a Lost Art

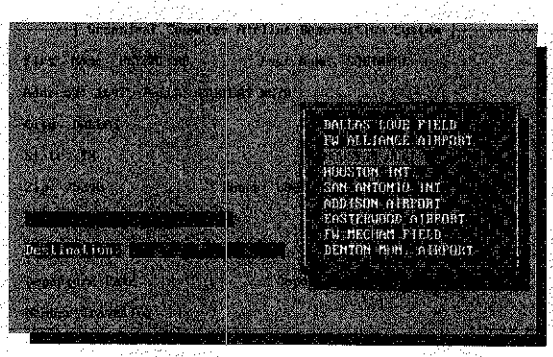
Enjoy a well-crafted user interface for C - one with all the details you need

Like the gears in a skillfully crafted timepiece, the low-level functions of Greenleaf DataWindows mesh and work in concert, hiding the complex details, to make your interface design easier!

Making an overlying window invisible and repainting the screen takes a single function call with DataWindows. Autodetection of display type, Enhanced Keyboard and mouse has never been easier. We do the grunt work so you don't have to!

Select from several styles of menus, including Pull-Down and Pop-Up, with infinite branching and runtime extensibility, 3 kinds of help and hundreds of other features!

Best of all a sophisticated data entry engine handles your database record as an entire transaction (unless all fields check out, the record isn't updated). Scrollable subforms, memo and tickertape fields, nested transactions, choice boxes, list boxes, 19 kinds of state-change function hooks for programmability - and so much more!



Forms larger than the display handled easily with auto or manual scrolling any direction.

Data entry and menus are based on logical windows which are special in DataWindows. The specialness comes from extra care at the lowest levels - so that you don't have to concern yourself with details like updating the screen! Save windows with text & attributes to disk.

User input from keyboard and mouse is monitored by a single call. Extras like input redirection to a menu or string, Ctrl-Break handling, and automatic display type sensing are built in.

Documentation includes two binders chock-full of data, tutorials, and compilable examples of each function, PLUS a complete online documentation system including a FREE PDQPlus™ engine! Other help engines supported too! A Greenleaf Exclusive!

Portability is assured by versions for DOS, UNIX, XENIX and VMS (including terminal definitions for multi-user systems).

So easy to use - yet jam-packed with extras. Used by professionals the world over.

CALL (800)523-9830 *UNIX, DOS, 386!*

FAX (214)248-7830

Greenleaf DataWindows v2.12 for DOS supports Microsoft C, Borland/Turbo C/C++, Watcom C, Zortech C++ and TopSpeed C . Versions for UNIX System V and SCO XENIX System V.

- ***Greenleaf MakeForm v2.0 Form Generator also available!***
- ***No royalties.***
- ***Source code is FREE!***
- ***Support is FREE!***
- ***Multi-User Extended Support is also available.***
- ***Free BBS and Newsletter!***
- ***Ask about Gold Support!***

GREENLEAF Software©

16479 Dallas Parkway, #570
Dallas, TX 75248
(214)248-2561

member TopSpeed® Consortium

Circle 7 on Reader Service Card

NEW PRODUCT!

DataWindows/386™

- ✓ Supports Phar Lap 386 DOS Extender
- ✓ Uses Watcom C 386 v8.0 or MetaWare High C v2.3
- ✓ All source code in ANSI C supplied FREE!
- ✓ Supports full 32-bit flat model for 386 or 486 systems.
- ✓ Logical windows — up to available memory — AND you never have to worry about repainting because DataWindows/386™ does that automatically — always!
- ✓ All features of DataWindows v2.12 (excepting DOS Redirection).
- ✓ Fully compatible with Greenleaf MakeForm v2.0 form generator.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

disks (but only if you were backing up DBF files).

Another programmer had a way to break any file across disks. Someone else had written a compression routine in assembly language that was linked to his Clipper code, but he didn't break files across disks. Many other ideas flowed from this, such as setting flags to determine backup frequency and warning the user periodically in the initialization module that backups were overdue.

The result, module by module, was an evolving set of specifications, even though people had been content before we started. When we finished, the users were happier with their systems.

The Japanese have a word for this process of continuous improvement. They call it *kaizen*. Even if you confuse the cry of suicidal warriors (*banzai*) with the name for dwarfed trees (*bonsai*), you ought to remember *kaizen*. What can you do to increase the quality of the systems that you write? Are your clients really happy, or is some firm going to woo them away with a better product?

INCH BY INCH

Measurement is another important concept in TQM. Without measurement, you can't tell whether you are improving. Don't be too quick to dismiss the feasibility of measuring the improvement of *your* particular business. Even though TQM did evolve in a manufacturing environment—where you can count the output, the number of defects, and the mean time between failures—the principle of measurement can be applied to any kind of work.

Let's say we run a hot line on our contract. Users call in and either we fix their problems over the phone or we dispatch other support teams to clean up the mess. An obvious measurement in this case would be how long it takes to close a call. If we find ways to improve our services—by getting answers to users faster, by dispatching teams from a more appropriately located organization, or by some other process—then we can expect the measure of response time to decrease. If it takes an average of four hours to close a call now and we cut that down to 17 minutes, then we can pat ourselves on the back for a tremendous improvement. We might eventually get that number down a few more minutes, but eventually we will hit a brick wall (and wish we had enrolled in that martial arts training instead). We'll have to look for other ways to improve, such as measuring the number of times we actually gave the *right* answers, or something revolutionary like that. One thing at a time.

As a practical matter, I think measurement may be one of those things that requires some substantial outside input. If you can't think of anything to measure in your own work, talk to your most vocal critic. Or better yet, go find your customer. I'm sure they'll come up with some dandy measures.

WE'RE ALL IN THIS TOGETHER

TQM bridges organization lines, and it even extends up and down the channels to your suppliers and your customers. If you're trying to make things work better, your customers must be a part of that process. How many times have you remarked that building custom systems would be a great business if it weren't for the users? Similarly, your work can't get done if your suppliers don't cooperate. (Suppliers can be outside vendors or internal suppliers of services, such as accounting or payroll.)

Now look at it from the point of view of the government (or whoever your client is): as *their* supplier, you may be part of *their* problem! If you can solve some of their problems by doing your job better, then you're going to have happier clients. What a concept.

I recall a similar principle from the world of MBAs—specifically, the retailing arena. If you sell hardware and software, then you're part of a retail channel that includes your software wholesalers and the publishers who supply them. You're at the end of the channel, supplying the customers. Similarly, you're at the end of a hardware channel that goes back to the manufacturer. In fact, you're at the end of multiple channels.

There are many other channels around, and they compete with yours. Your software wholesaler probably sells to other shops (creating competing channels), and you probably buy from several software wholesalers. It's in your best interest to work with the other members of your channel to deliver the goods better than any other channel. That might mean having software publishers come to your store to train your people, so that you can offer better service than the gas station on the corner that sells corn chips and spreadsheets. The wholesaler may have training arrangements, too. An enlightened publisher will realize that if the customers from the chain of gas stations comprise 1 percent of their sales and 90 percent of their support calls, then maybe there's a better way to do business.

The point is, if you don't work both with your suppliers and with your customers to find out what you can do better, someone in another channel entirely will come along and deliver a better product or service. I hate it when that happens.

TURNING THE CHANNEL

If you're in the consulting business and you use third-party libraries, consider offering the manufacturer some suggestions of how they could improve their product. Smart publishers are delighted to have these suggestions; they want to work with their customers, because they know that their success depends on the success of the consultants using their products. For example, if you do Clipper work, you're in a channel partnership with Nantucket and the third-party publishers. This channel is competing with mainframe, mini, and other micro channels such as the Paradox consultants. Join a local Clipper developers' group. Although you may think of the other members as your competitors, all of you are competing against HP VAR channels, Rbase channels, and others.

Help your channel members, all the way up to Nantucket. Nantucket, to continue the example, is the quintessential nonchannel player. They nominally support the user and developer groups, but the groups don't feel warm and cuddly about Nantucket because Nantucket doesn't understand their role as part of a larger channel. If they're not careful, the Fox channel is going to blow them away.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A TQM BY ANY OTHER NAME

One of the best things about TQM is that you can play around with the concepts, putting them together in different combinations and even inventing new acronyms. In a discussion group the other day, a colleague came up with the acronym "RAT" to illustrate three related ideas.

The *R* stands for responsibility. Each person has to take responsibility for the quality of their work when you buy into TQM. You can't just shrug something off as "not my job." You can see why TQM needs to become an attitude to be effective; responsibility is easy to talk about, but getting your team to buy into it is more difficult.

The *A* stands for authority. If you take responsibility for your work (or worse, if you're held accountable against your will) but you don't have any control, you'll quickly go mad. TQM suggests pushing the authority down as far as it can go. This would mean that each worker on the assembly line would have the authority to reject a part if it didn't meet standards. Each worker has the authority to back up the responsibility that they're asked to take on.

Most people don't know what they're doing half the time; before they get it figured out, someone transfers them or promotes them to a new job.

As another example, consider how raises are handled in your company. Traditionally, lower-level managers may suggest a percentage, but high-level managers make the decision. TQM suggests that the top-level management push this process down: set the raise pool at a high level, for example, and then tell the next management level down that they have *x* number of dollars to hand out. This process can go on until the people handing out the money are the direct supervisors of the people getting the raises.

The *T* stands for training. You wouldn't ask someone to dig a ditch without a shovel, so why would you take someone who was hired to do office automation support, give them a manual, and ask them to develop systems? Don't underestimate the value of training. Most people don't know what they're doing half the time; before they get it figured out, someone transfers them or promotes them to a new job. I've seen many programmers who can't write decent code, but they can screw up in five languages. Their COBOL is cobbled, their Oracle orbicular, their

INSTALL.BAT

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

First impressions last.

INSTALL 3.0 gives your software product a professional introduction.

Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.

INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.

- Features very sophisticated error-handling
- Does high performance data compression
- Can safely and reliably modify system files
- Allows selective installation of parts of your program
- Complete source code included
- No royalty charges
- Free technical support

INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.

INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.

KNOWLEDGE DYNAMICS CORP.
P.O. Box 1558
Canyon Lake, TX 78130-1558
MasterCard/VISA/COD/POs welcome

$249.95 + shipping
Money back guarantee
Sales 800-331-2783 Ext 0196
Internat'l 512-964-3994
24 hr FAX 512-964-3958
24 hr BBS 512-964-3929

Circle 10 on Reader Service Card

Instant Windows

Gorgeous Screens on Windows,DOS,Unix...

Rapidly Prototype applications in a few hours. You can link Windows,Dialogs,Pushbuttons and Menus. Animate,test and demo look-and-feel without coding. A point-and-click tool lets you design Fields,Help,Editors, **Spreadsheet-like Tables**,Icons,Bitmaps,Directories, radio/check boxes,push buttons,list/combo boxes...

Data Validations in fields, forms and tables,e.g.: Custom validations,cross,protected,required, lookups, range,date,time,money,lists,pictures(A9!LXB#Z$).....

Quality C Code Generated
Connect to LANs,SQL,3270/Hosts,Sybase,Informix, CTree, Oracle,BTrieve, dBVista, Paradox,dBase. You can easily add in your own logic, modify and regenerate the code.

Port your application in hours...
Windows $495.DOS $249.OS/2 PM $995.UNIX/Mac Call.

800-334-9552 Ph. 415-324-9552 FAX 415-324-9580

WinSoft. 1016 E. El Camino Real#216,Sunnyvale CA 94087

Circle 11 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dBASE defiled, their Paradox paranormal, and their Fortran is all forked up. What they need is time out to learn how to code well in one language, how to structure code, how to debug it, how to document it, and how to prevent maintenance nightmares for other people.

FULL OF SURPRISES

I was cynical about whether TQM would work in a traditional corporate environment. First, I knew that the process was probably client-driven: the government wants it and wants to see the contractors embrace it, so therefore we want it. QED.

Second, for every principle of TQM that I've heard espoused, I can think of dozens of high-level managers who embody the antithesis. You have to figure that there's going to be some powerful opposition to TQM from those at the top, which is precisely where support for it has to come from.

But in spite of the initial cynicism and the resistance of inertia, it looks as if TQM is starting to work. The folks who hang out above the cloud cover on the top floor keep coming up with new ideas, such as salary pools. Naturally, my peers and I think these ideas are great, and we think we've been practicing most of them all along. But when TQM gets to the level below us, we may be in for a rude awakening. After all, the top-level managers were pretty surprised at some of the problems that *we* brought up: they thought everything had been working just fine.

In spite of the initial cynicism and the resistance of inertia, it looks as if TQM is starting to work.

GETTING BETTER ALL THE TIME

Three things seem to come out of TQM. The first thing has nothing to do with the merits of the concept. If you made up any acronym, say PAB, and then got everyone together, starting at the top, to talk about problems and how they could be solved, you would be bound to see improvement. Experimental psychology shows that almost anything you do in a positive way will improve morale and output. I know PAB can make a big difference in your company; now I'm looking for a foreign country that will make me a cultural idol, preferably one where the climate is mild, English is spoken, and the food is decent.

A second benefit of TQM is personal improvement. As each person increases his or her skills through training, the worth of each individual increases.

OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

OPUS Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest, most full-featured make utility there is. Features include:

- DOS version uses only 3K memory!
- Multiple directory support.
- Supports: Polytron PVCS™, Burton Systems TLIB™, Microsoft LIB™, SLR Systems OPTLIB™
- Generates automatic response files of unlimited length for LINK and LIB.
- OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or P.O. 60 day MBG!

Call 1-800-248-OPUS
International: 415-924-7801
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 12 on Reader Service Card

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3
Excel • Paradox • Informix • Q&A • Oracle • Clarion
DataFlex • askSam • SC4 • Xdb • DataEase • Symphony
ACT! • Platinum • DAC • ACCPAC+ and other apps
Btrieve • c-tree • EBCDIC • Packed & Zoned decimal
IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries & Floats

Filter Your Files While Converting!
- Menu driven — No programming
- Select & rearrange records/fields/chars
- Instant scripting for batch converts
- Written in C — Ideal for OEM apps

Clean Up & Massage Your Data!
- Merge & split fields; Name/address parsing
- Search/replace and case translation by column
- Edit/enter corrections — Build custom data types

1990 DATA BASED ADVISOR The Database Management Systems Magazine Readers Choice Award

"... an invaluable integration tool." —Byte
"... more than worth its price." —PC Magazine
"... a surefire winner." —Computer Language

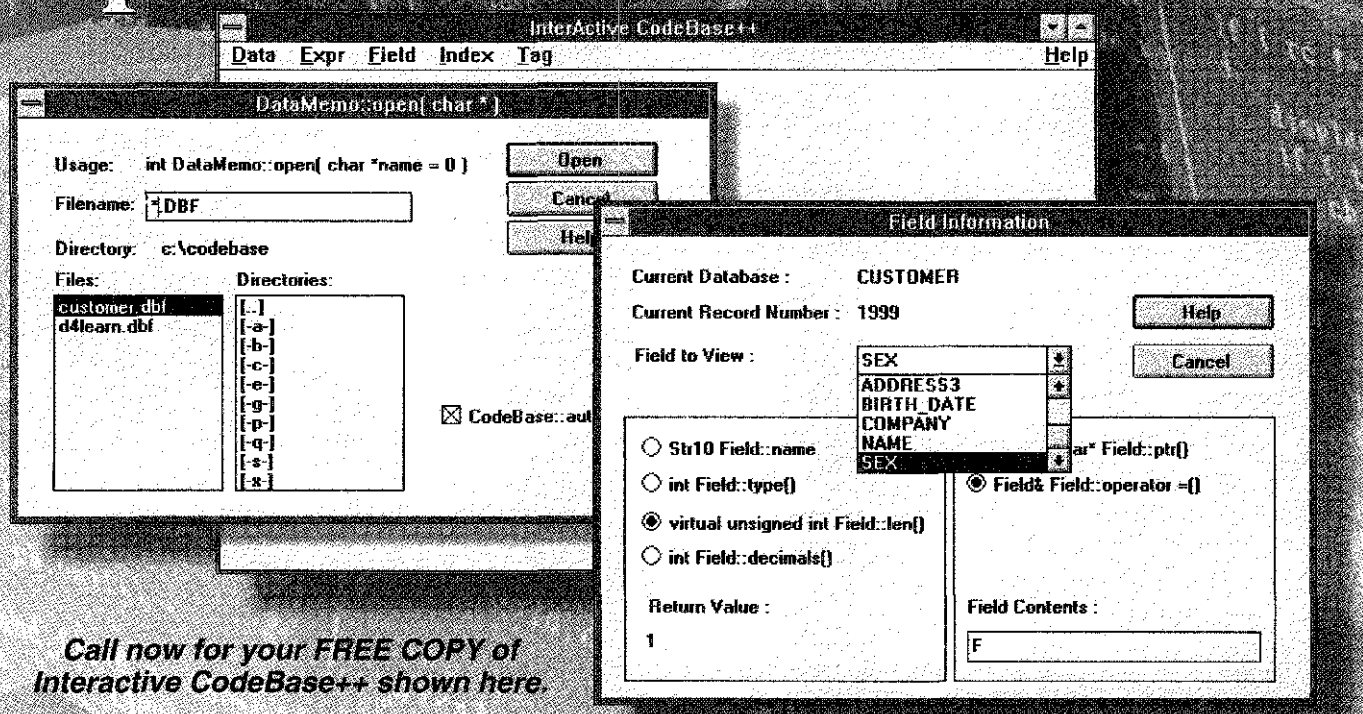
Standard $99 — Converts dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam, & merge files for WordPerfect, Word & Wordstar
Professional $199 — Standard formats plus Binary/EBCDIC, SC4, SYLK, Paradox, R:BASE, Clarion, Excel, DataEase, DataFlex, c-tree, Btrieve
Advanced $299 — Standard & Prof. formats + Informix, Xdb, Oracle, C-ISAM

Call 800/444-1945 or 512/482-0824 • FAX 512/482-0976
Also available for LANs, Xenix & Unix. MC/VISA • AMEX • COD • Corp. PO.
Tools & Techniques Inc • 1620 W 12 • Austin TX 78703

Circle 13 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Why do dBASE professionals use C++?



Call now for your FREE COPY of Interactive CodeBase++ shown here.

• dBASE Functionality

Use CodeBase++, a C++ class library, for database management.

• Object Oriented Syntax

dBASE Code

```
* Assign to field LAST_NAME
LAST_NAME = "Jones"
```

CodeBase++ Code

```
// Assign to field LAST_NAME
LAST_NAME = "Jones" ;
```

• Low Memory Requirements

A CodeBase++ .EXE file is as small as 20k.

• Speed

"CodeBase++ is fast: I indexed a .dbf of 10,000 records in 32 seconds; dBASE IV took 40 seconds." Kathy Cea, DBMS

• Portability

CodeBase++ ships with a Microsoft Windows 3.0 DLL.

• Compatibility

Codebase++ is completely compatible with dBASE and Clipper data and index files.

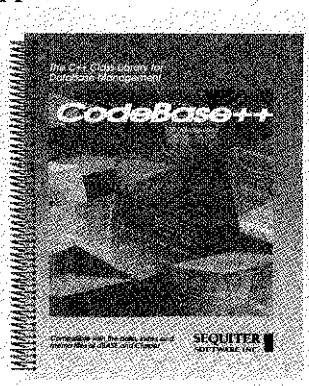
CodeBase++

The Power of dBASE, the Versatility of C++

SEQUITER SOFTWARE INC.

TEL. 403•437•2410
FAX 403•436•2999

#209,9644-54 AVE., EDMONTON, AB, CANADA T6E-5V1



Circle 14 on Reader Service Card

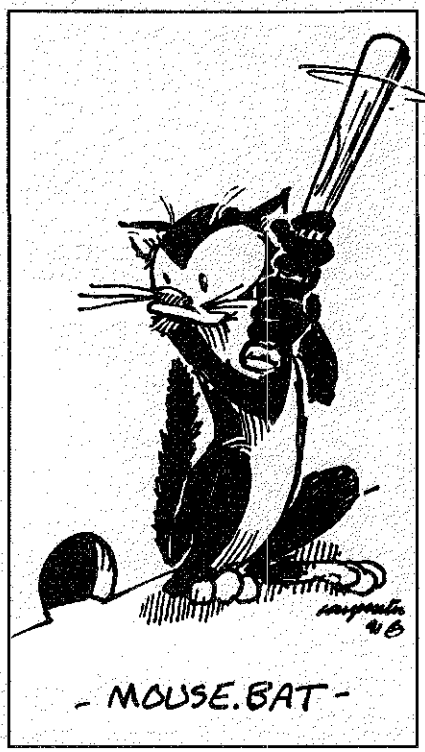
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

From a purely selfish point of view, TQM is worth doing because it makes you a better (as in more highly skilled) worker.

TQM's third benefit is that it improves the quality of life. When everyone works together to solve problems across the organization, routine screw-ups begin to decrease. Open communication reduces the problems attributable to lack of understanding. When you stop operating in a fire-fighting mode and start taking charge of your portion of the consulting business, you'll find that your work week gets a little more pleasant, and a little shorter, too. If the negative aspects of your job are affecting your personal life, you can expect some gradual changes there, as well. I haven't heard anyone say that TQM will improve your love life, but hey, it can't hurt.

If TQM sounds like it might help you with your practice, I suggest investing some serious effort in learning more about it. A one-day seminar might cover most of the concepts and explain them briefly, but assimilating TQM requires more work than that. You've got to really embrace it, and you've got to get everyone else in your company and preferably among your suppliers and customers to buy into it as well. The payoff can be enormous, but plan on many focused discussion sessions and applied exercises such as Process Improvement Teams.

Can the concepts of TQM actually affect the federal government? Consider this parallel phenomenon: at one time the label "Made in Japan" was synonymous with junk, but today, that same phrase means quality. If TQM can be instrumental in such a flip-flop, maybe some day it can change the status of the phrase "I'm from the government and I'm here to help you." Instead of being thought of as one of the top three archetypal lies, it might actually be taken seriously. ■



Tired of the CONFIG.SYS shuffle?

Let BOOTCON simplify your life.

"... it's bulletproof ... The interface is well thought out. You need this program."
Dr. Jerry Pournelle, *Byte*, July 1990

"Highly recommended."
John C. Dvorak, *PC Magazine*, July 1990

How many CONFIG.SYS and AUTOEXEC.BAT combinations do you have on your PC? BOOTCON eliminates the confusion by storing all your configuration information in just one set of configuration files. *Each time you boot your PC* you can select the configuration you want to use from a menu of up to 26 choices.

BOOTCON is ideal when you need to use conflicting programs such as Windows 3.0, QEMM/386, 386MAX, Soft-ICE, and Lotus 123. A default configuration will automatically be used if you do not make a selection within the specified time-out period. BOOTCON also provides an optional password protection feature.

BOOTCON lists for $59.95 and comes on both 5¼" and 3½" disks. It has an unconditional 30-day money-back guarantee. MS-DOS or PC-DOS 3.10 or later is required.

Modular Software Systems

115 West California Blvd., Suite 113
Pasadena, CA 91105

Phone (818) 440-9104 FAX (818) 440-9240

Windows and MS-DOS are trademarks of Microsoft Corporation. PC-DOS is a trademark of International Business Machines Corporation. QEMM/386 is a trademark of Quarterdeck Office Systems. 386MAX is a trademark of Qualitas Corporation. Soft-ICE is a trademark of Nu-Mega Technologies. Lotus and 123 are trademarks of Lotus Development Corporation.

Circle 15 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Chips off the old block

By Dave Thompson

So you'd like a pen-based system, or maybe you're auditory and you're interested in voice recognition, or maybe you do a lot of compiles and you'd like to see a thousand-dollar clone that runs 50 MIPS, or how about a compiler that fixes your mistakes rather than yelling "Syntax Error"?

You can start the ball rolling by simply writing a press release. For instance, you could announce that you're "BMI Corp." and you'll be coming out with a thousand-dollar CRAY XT during the fourth quarter of 1993.

Now, send the release to a bunch of magazines. (Be sure to send me a copy; see below.) Chances are good that someone will run it. Then IBM, DEC, Microsoft, Borland—all those folks who watch new product announcements—will get excited, and they'll have to get cracking so they don't get left out and . . . Well, you get the picture.

NO CHIPS AT ALL?

Motorola just sent out a notice about their new PC-compatible floppy controller. The MCS3201 chip handles as many as four drives in any mix of 360K, 720K, 1.2MB, or 1.44MB. Motorola has included address decoding, variable-write precomp, data scan over a single sector or whole track, and DMA/non-DMA transfers. All you supply is 5V and a crystal. (Chips run $5 each in OEM quantities.)

Meanwhile, Elite's Eagle chip set, which supplies most of the glue for a 386DX, makes possible a complete cache-based motherboard with as few as 25 components. Now just add Motorola's floppy IC, a crystal, and presto zippo!

Just wait, though. When they start putting RAM on board the support chips we'll be seeing complete systems in

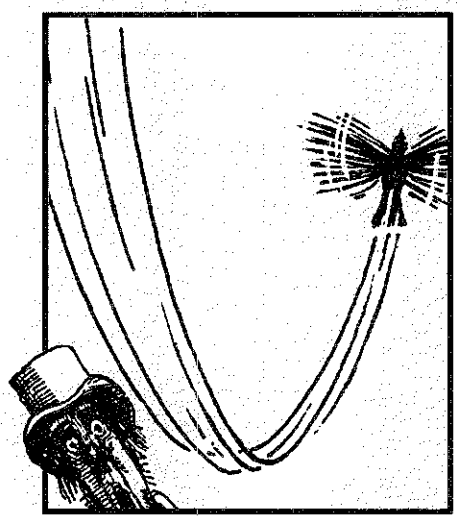


ILLUSTRATION BY JIM CARPENTER

three or four ICs. Make them out of CMOS and, who knows, you could have solar-powered, lap-loving screamers.

Or you could wait for Intel to come out with its next-generation 586 or whatever. They're supposed to be working on a single-chip system, glue and everything, in 300 pins or less.

OPENING MY MAIL

While roaming loose in the *PJ* offices recently, I happened upon the following piece of mail:

Dear Dave,

To our disappointment, you omitted mentioning our company's Siege and Besiege product lines. We're Mockingbird Communications, and our products are considered the best on the market in terms of performance, maturity, and user friendliness.

As you will be describing products like Siege and Beseige in an upcoming issue, I would like to take this opportunity to provide you with whatever information you require in order to include us in your article.

Yours truly,

I.M. Adement

P.S. If you run something about us in the next issue, you can ignore the two heavyset fellows stationed in the Mercedes across the street from the PJ offices.

P.P.S. If the mention is particularly positive, you can keep the Mercedes.

To which I made the following reply:

Dear I.M.,

Thanks for your letter. I'm sure readers would like to know more about your company, since they spend their lives working under Siege. Have you considered exporting your product to areas where it's truly needed? (The Middle East, Eastern Europe, New York City . . .)

LOOK AND FEEL OF FORTAN

PJ received another letter saying something about maintaining the look and feel of Fortran. Give me a break. I spent months in school solving calculus in Fortran (it was all we had). Assembly language would have been easier—it's more like English.

I prefer Pascal. It's like pseudo code. Heck, by the time I've got a problem written and debugged in Pascal I've come up with 20 additional features, all of which must be added. It's slow but it's fun, and I can spend months elaborating on something as simple as "Hello World." (For instance, this would be a great program for playing with compression algorithms.)

And speaking of playing, try C'ing for the first time, or after not writing C for a while (like half an hour). Wow! Just keeping the compiler from croaking is a challenge (and the comments you get from lint!). You can have so much fun with C that you forget why you're using it.

SON OF TECHTIPS

Those who remember *Micro Cornucopia* will remember Techtips, its most popular column. Readers sent in tips, and we printed the gems. You'll be delighted to know that *PJ* is now continuing this proud old tradition.

Tips can include algorithms, hardware fixes and breaks, cable diagrams, where to get parts or special kinds of software, interrupt vectors or other operating system insides, I/O interface hardware, and anything else of practical value that can be described in three

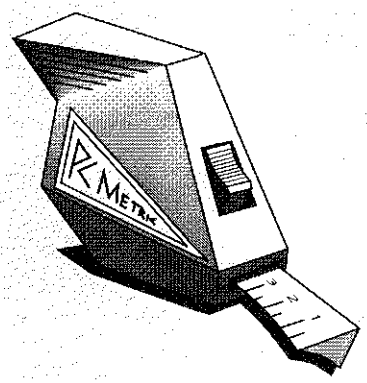
---

Dave Thompson is a former hardware/software engineer at Tektronix and the past editor of the prehistoric programming magazine Micro Cornucopia. *He can be reached at P.O. Box 223, Bend, OR 97709.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Measure of a Great Program.

PC-METRIC™: The Measurement Tool For Serious Developers.



PC-METRIC is the software measurement tool that measures your code and identifies its most complex parts. So you can spend your time working in the areas most likely to cause problems.

PC-METRIC is fast, user-configurable, and includes a wide array of commonly accepted measurement standards.

Plus, versions of PC-METRIC are available to support virtually every popular programming language.

A Great Value By Any Measure.

PC-METRIC's price is only $199, and it comes with a 30-day money-back guarantee. Multiple user discounts are available, as well as site licenses and complete source code.

Order Now! Call (503) 829-7123.

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

SET LABORATORIES, INC.
"Quality Tools For Software Craftsmen"
P.O. Box 868
Mulino, OR 97042
Phone: (503) 829-7123
FAX: (503) 829-7220

Contact this advertiser directly.

YOU OUGHT TO BE IN ☆REPRINTS☆

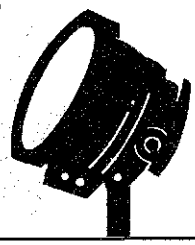
You're the picture of success.
And **PJ** reprints are a great way to showcase your four-star performance. Reprints help you:

- ☆ Enhance your trade show presentations
- ☆ Promote your products and services
- ☆ Increase your marketplace presence
- ☆ Optimize your product training sessions

PROGRAMMER'S JOURNAL offers full-color or black-and-white reprints.

PJ reprints keep you in the spotlight.

Call us today at 800-234-0386.



paragraphs or less and/or 20 lines of code. You can even throw in a very small schematic.

Write each tip on a separate piece of paper. Include your name, address, and phone number. Then send them to me at *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401-0110.

Authors will become famous, of course, and they'll also get an exclusive, authors-only T-shirt. At *Micro C*, we kept these highly coveted shirts in a plain, unmarked box concealed behind the renewal notices in the shipping department. Our bank refused to let us keep them in the safe deposit box after they learned that our readers were capable of waylaying electronic transfers. Wearers have even been offered bribes to sell the shirts off their backs.

So if you'd like to get a special package from *PJ*, all you have to do is send in one of those "Hey, did you know that with this little bit of code you can . . ." You don't even have to have made it up—just be the first to send it in. (Plus, your Techtip might supply a future "Programmer Bob" contestant with a great idea—see p. 49.)

WHY DON'T YOU WRITE ME

Now back to those press releases: if you come up with a product suggestion—funny or straight—or if you write or receive a particularly, shall we say, *interesting* press release, send that to me too, using the *PJ* post office box given above. (If you think I'm hungry for mail out here, you're absolutely right.) If it makes me laugh, I'll mention it.

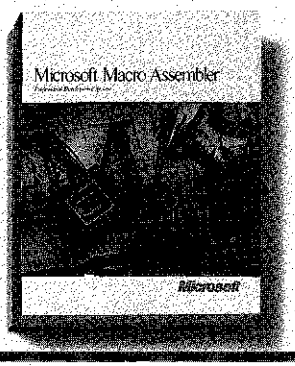
The oxymoron for this month: Safe pesticide. PJ

VENDOR INFORMATION

Motorola Inc.
MOS IC Division
P.O. Box 6000
Austin, TX 78762
(512) 928-6880
Circle 211 on Reader Service Card.

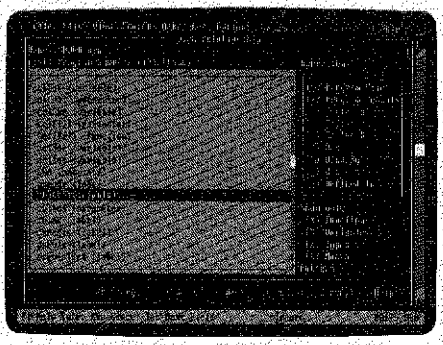
Elite Microelectronics
4003 North First Street
San Jose, CA 95134
(408) 943-0500
408-943-0561 (FAX)
Circle 212 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



MICROSOFT SOFTWARE DEVELOPMENT TOOLS

Microsoft MASM 6.0. Now 23% less painful.



Source Browser lets you explore the relationships between functions, variables, types, and macros in MASM-only and mixed-language programs.

Microsoft CodeView gives you instant access to all of your source and data, including memory, registers, and flags. You can also access Microsoft Professional Advisor online documentation to get instantaneous assistance.

We know exactly how you feel about assembler: Low-level language, high-level frustration.

But the new Microsoft® MASM 6.0 relieves the stress and strain of Assembler programming (exactly how much depends on how crazed it made you in the first place).

MASM 6.0 has additional high-level language extensions for easier programming. It has the ability to convert C header files into MASM include files, so you don't have to. And the intelligence to optimize the shorter jumps and eliminate NOPs and phase errors from the longer ones.

You also get 386™/486™ instruction set support.

And a Programmer's WorkBench that supports development of MASM-only or mixed-language programs. All under one seamlessly integrated environment.

So call (800) 541-1261, Dept. Q59, for a free white paper. And fast relief from headache pain.

For more information inside the 50 United States, call (800) 541-1261, Dept. Q59. Customers in Canada, call (416) 568-3503. Outside the U.S. and Canada, call (206) 936-8661. *© 1991 Microsoft Corporation. All rights reserved. Microsoft, the Microsoft logo and CodeView are registered trademarks and* Making it all make sense *is a trademark of Microsoft Corporation. 386 and 486 are trademarks of Intel Corporation.*

New Features

- High-level language data types and control structures make mixed language programming easier. Including: PROTO/INVOKE, TYPEDEF, .IF/.THEN/.ELSE, and .REPEAT/.WHILE.
- Macro language extensions such as: @SubStr, @CatStr, @Date, & @Time.
- H2INC converts C header files to Assembler include files.
- Automated Jump shortening speeds execution.
- 486 and expanded 386 instruction set support with flat model (32-bit) support.
- *n*-pass Assembler averts phase errors and Assembler-generated NOPs.
- User-defined Prologue/Epilogue customizeable for Windows.
- /Zm switch plus other features to help accommodate your MASM 5.1 code.
- Significantly improved Assembler speed.
- Programmer's WorkBench v 1.1 and CodeView® v 3.12 make it easier to develop stand-alone .asm and mixed-language programs.
- Source Browser gives instant access to data declarations and function definitions.

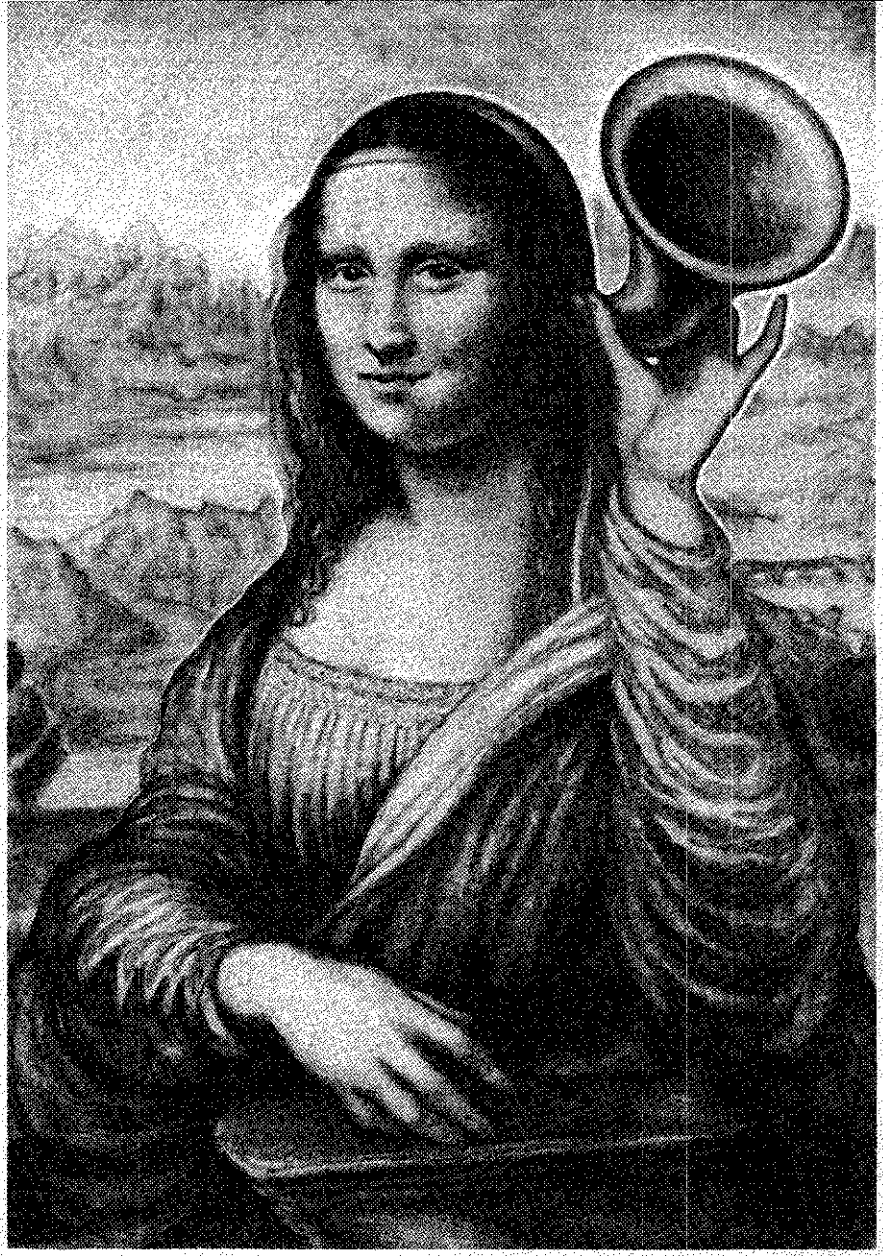
PROGRAMMER'S TIPS

The new INVOKE statement automates procedure calling, by handling all pushing and popping of parameters on the stack. Example:
INVOKE printf, ADDR fmt1st,
ADDR str2, dat1, dat2
If you want MASM 5.1 compatibility, start with the MASM.EXE compatibility driver or use the /Zm switch with ML.EXE.

Microsoft
Making it all make sense®

Contact this advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Multitasking operating systems thrive on the 80386 processor's built-in support for task switching and protection. Aspects of this protection include the hardware verification of a task's memory accesses and its optional accessing of I/O addresses. This latter capability lends itself perfectly to monitoring a task's I/O activity. Trapping and logging I/O accesses can be useful for debugging, black boxing, and educational and entertainment purposes.

This article describes IOM.EXE, an I/O monitoring program for the MS-DOS environment. To implement this task-protection feature, the program uses the 80386's protected and virtual-8086 modes. (See "Running unprotected," by Dean Wills, in *PJ* 8.4.)

Let's begin by reviewing the main data structure that supports multitasking.

LIMITED ACCESS

In 80386 protected mode, the task state segment (TSS) is the data area that holds the context of a task and static information, including other privilege-level stack segments and pointers and an optional I/O permission bitmap. This bitmap is what specifies which I/O addresses a task can access.

The I/O permission bitmap contains one bit for each address in the processor's I/O space. Because I/O addresses range from 0 to 65535, the bitmap can be as large as 8K. However, it only needs to be big enough to cover the maximum I/O address being tested. For example, if the highest I/O address to be tested is 119 (decimal), the bitmap only needs to be 16 bytes long (including the pad byte that must be placed at the end).

The processor locates the I/O permission map by adding the offset that is contained in the TSS's I/O map base field to the address at which TSS starts. It calculates the length of the I/O permission map by subtracting the I/O permission map offset from the limit of the TSS specified in the global descriptor table (GDT) entry for the TSS segment.

In protected mode, if a task's current privilege level (CPL) is less privileged (is numerically higher) than its I/O privilege level (IOPL), the I/O permission map is consulted to determine whether the task may access the particular I/O address.

In virtual-8086 mode, the processor consults the map regardless of the IOPL. If the corresponding permission map bit is set, the task cannot access the port. In such cases, a general protection exception (interrupt 0Dh) is generated. This exception is a fault; it enters the exception handler with the CS:EIP of the instruction as part of the information on the stack.

AN EYE ON I/O

The I/O monitoring program requires a number of steps. First, put the processor into protected mode and initialize all required areas; this includes setting bits in the TSS I/O permission bitmap for the I/O addresses that you wish to monitor. Second, create and execute a virtual-8086 task. This provides real-mode compatibility, allowing DOS and any applications to run within the restrictions that you impose via IOPL and the TSS.

When I/O accesses involving the specified range of ports are attempted, control is transferred to the interrupt 0Dh handler. If the instruction causing the exception is an I/O instruction, the I/O monitor temporarily clears the bits that correspond to the port(s) in the I/O permission bitmap; then it single steps through the instruction, stores the input or output data, and again sets the bits in the I/O permission bitmap.

Let's look at each of these steps in more detail.

In protected mode, segment registers are selectors that contain indices into

---

Rick Knoblaugh is a software engineer specializing in systems programming for microcomputers. He is a co-author of Screen Machine, a screen design/prototyping/code generation utility. He can be reached at P.O. Box 1109, Half Moon Bay, CA 94019.

ILLUSTRATION BY BRENT BURKETT

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Eaves-dropping on I/O

Trapping and monitoring I/O port use on the 80386

By Rick Knoblaugh

either the global descriptor table (GDT) or the local descriptor table (LDT). Entries must be created in the GDT for each segment of memory to be used. These entries define the segment's size, type (executable, read only, or read/write), and other attributes.

Another difference between real and protected mode is that in protected mode, interrupts pass through gates in the interrupt descriptor table (IDT) rather than through the interrupt vectors. Thus, IDT entries are provided that point to handlers in the I/O monitor. These handlers either perform special interrupt processing or simply route the interrupts to the original interrupt service routines (via the real-mode interrupt vector table).

INTERRUPT HANDLING

In the PC environment, one complication of protected-mode programming is that certain protected-mode processor exceptions use the same interrupts as the hardware interrupts generated from IRQ0 to IRQ7. For example, IRQ5—typically used with LPT2—generates interrupt 0Dh. Note that 0Dh is the interrupt generated for the general protection exception discussed above. To avoid this conflict, I reprogrammed the master PIC so that interrupts start at 20h rather than at 08h.

Because V86 tasks execute with CPL = 3 and because we have specified IOPL = 3, *sensitive* instructions that require (CPL <= IOPL) can be executed. These instructions include I/O instructions and those, such as INT *n*, that directly affect the interrupt flag. Software interrupts do not go directly through the IDT gates, however, because all IDT entries have been defined with descriptor privilege level (DPL) set to zero. Thus, all software interrupts will cause a general protection exception (interrupt 0Dh). When the interrupt 0Dh handler detects this, it transfers control to the appropriate software interrupt routine.

After the GDT and IDT are established and loaded, you switch the processor to protected mode, load the task register with the index into the GDT for the TSS, and initialize the LDT to NULL (because the task will not make use of the LDT). Virtual-8086 mode is then entered by building a protected-mode interrupt stack frame, specifying that the VM bit in the EFLAGS be set (and that IOPL = 3) and setting CS:EIP so that the IRETD causes a transfer of control to the next instruction in the program. The code will execute as the virtual-8086 task. The I/O monitor then terminates and stays resident in your system.

DETECTING I/O

Interrupt 0Dh occurs when DOS, the BIOS, or other code attempts to execute an I/O instruction that accesses a monitored I/O port. The interrupt 0Dh handler then performs a table lookup on the instruction that caused the exception. If the instruction is an I/O instruction (supported by the table), the port address is stored and the corresponding TSS I/O permission map bits are cleared to allow the instruction to proceed. The trap bit is set in the EFLAGS so that a single-step trap (interrupt 1) will be generated after the I/O instruction executes. The I/O monitor interrupt 1 handler then records the input/output data and sets the I/O permission map bits again.

To demonstrate this technique, I have provided support in the code and instruction look-up table for basic 8- and 16-bit **in** and **out** instructions. To keep things simple, support for 32-bit I/O and I/O string instructions has been omitted.

Note that if these instructions are executed within the specified range to be monitored, the I/O monitor will display the address of the instruction and then stop. Monitoring these other instructions could be added with a little work.

The I/O monitor is invoked in the following way:

```
IOM startrange endrange
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
;Listing 1 contains a number of different files. Although they are separate
;on the code disk, they have been merged here for convenience.-- Ed.
;
;iomon.inc - equates for I/O Monitor
;
NULL            EQU     0
TRUE            EQU     -1
FALSE           EQU     NOT TRUE
BUF_SIZE        EQU     4096
DOS_PRT_STRING  EQU     9
DOS_TSR_FUNC    EQU     31h
OK              EQU     0
BEEP            EQU     7
CR              EQU     13
LF              EQU     10
TAB             EQU     9

USER_INT        EQU     61h             ;change this to another if it conflicts
PRESENT         EQU     80H
DPL0            EQU     0
DPL3            EQU     3
INT_GATE        EQU     0eh             ;80386 int gate
RW_DATA         EQU     92h             ;read/write data
ER_CODE         EQU     9ah             ;execute/read code
TSS_DESC        EQU     89h             ;Task State Segment descriptor
IO_MAP_SIZE     EQU     8192
TRAP_FLAG       EQU     100h            ;bits in eflags
INT_FLAG        EQU     200h
NT_FLAG         EQU     4000h
GEN_PROT_EXCEP  EQU     0dh             ;General Protection int number

INPUT           EQU     0               ;used in io_table to describe
OUTPUT          EQU     10h             ;the various I/O instructions
ABYTE           EQU     1
AWORD           EQU     2
CONSTANT        EQU     8

CONTAINS_DATA   EQU     80h
INT_OPCODE      EQU     0cdh
INT3_OPCODE     EQU     0cch
VID_PAGE_SIZE   EQU     4000
EXPECT_INT1     EQU     1
;
;macros for I/O monitor
DOINT           macro   num
int_&num        label   near
                PUBLIC  int_&num
                push    bp
                push    &num
                jmp     pass_thru
                endm

DOEXCP          macro   num
except_&num     label   near
                PUBLIC  except_&num
                push    bp
                push    &num
                jmp     except_handler
                endm

DOEXCPH         macro   num
int_&num        label   near
                PUBLIC  int_&num
                push    bp
                push    &num - 18h
                jmp     pass_thru
                endm

DEFIDT1         macro   num
                idt     <isrcode:int_&num,gdt_seg:sel_isrcode,,>
                endm

DEFIDTE         macro   num
                idt     <isrcode:except_&num,gdt_seg:sel_isrcode,,>
                endm
;
;structures for I/O monitor
idt             struc
i_offset        dw      ?
i_selector      dw      ?
i_unused        db      0
i_dpl_id        db      PRESENT + (DPL0 shl 5) + INT_GATE
i_offset2       dw      0
idt             ends

seg_descrip     struc
seg_limit_low   dw      ?
seg_base_low    dw      ?
seg_base_mid    db      ?
seg_type_dpl    db      ?
seg_limit_gran  db      ?
seg_base_top    db      ?
seg_descrip     ends

err_stack_area  struc                   ;stack with error code
e_pushed_int    dw      ?               ;int pushed by I/O monitor
e_pushed_bp     dw      ?               ;bp pushed by I/O monitor
e_errcode       dd      ?               ;error code
e_eip           dd      ?
e_cs            dw      2 dup(?)
e_eflags        dd      ?
e_esp           dd      ?
e_ss            dw      2 dup(?)
e_es            dw      2 dup(?)
e_ds            dw      2 dup(?)
e_fs            dw      2 dup(?)
e_gs            dw      2 dup(?)
err_stack_area  ends

stack_area      struc                   ;stack without error code
s_pushed_int    dw      ?               ;int pushed by I/O monitor
s_pushed_bp     dw      ?               ;bp pushed by I/O monitor
s_eip           dd      ?
s_cs            dw      2 dup(?)
s_eflags        dd      ?
s_esp           dd      ?
s_ss            dw      2 dup(?)
s_es            dw      2 dup(?)
s_ds            dw      2 dup(?)
s_fs            dw      2 dup(?)
s_gs            dw      2 dup(?)
stack_area      ends

user_stack      struc
user_ip         dw      ?
user_cs         dw      ?
user_flags      dw      ?
user_stack      ends

psp             struc
                db      80h dup(?)
arg_len         db      ?
arg_off         db      ?
psp             ends

doub_word       struc
d_offset        dw      ?
d_segment       dw      ?
doub_word       ends

io_struc        struc
io_opcode       db      ?
io_info         db      ?
io_struc        ends

buf_record      struc                   ;format of logged data
buf_info        db      ?
buf_port        dw      ?
buf_data        dw      ?
buf_record      ends

tss_dat         struc
t_backlink      dw      2 dup(?)
t_esp0          dd      ?
t_ess0          dw      2 dup(?)
t_esp1          dd      ?
t_ess1          dw      2 dup(?)
t_esp2          dd      ?
t_ess2          dw      2 dup(?)
t_cr3           dd      ?
t_eip           dd      ?
t_eflags        dd      ?
t_eax           dd      ?
t_ecx           dd      ?
t_edx           dd      ?
t_ebx           dd      ?
t_esp           dd      ?
t_ebp           dd      ?
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

where **startrange** and **endrange** are the starting and ending addresses (in hex) of the range of I/O ports to be monitored. Both must be specified.

TIME TO GET TO WORK

For example, to monitor I/O activity to LPT1, you would enter

```
IOM 378 37f
```

You shouldn't specify a broad range or one that includes a port that is accessed constantly, such as 20h or A0h: the I/O monitor round robin storage buffer would be filled instantly.

To retrieve a pointer to the I/O monitor storage buffer, execute user interrupt 61h. You can change this to another interrupt if it competes with some other program. A far pointer to the buffer will be returned in DX:BX. The code disk includes IODISP.C, which demonstrates retrieving the data.

You shouldn't specify a broad range to be monitored, or one that includes a port that is accessed constantly, such as 20h or A0h.

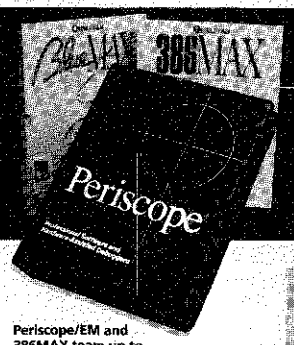
WORDS TO THE WISE

Because real-mode software debuggers can't be used, debugging protected-mode programs such as this I/O monitor can be a challenge. If, like me, you don't have an in-circuit emulator, you will need to be patient and reset your machine often.

I hope the template for protected-mode initialization given here is useful. When employing this code for other protected-mode applications, be sure to enable the A20 line if you intend to address more than 1MB. You might also want to investigate INT 15h function 89h, which can be used for much of the initialization. PJ

PERISCOPE

Professional Software and Hardware-Assisted Debuggers for 80386 & 80486 Systems



Periscope/EM and 386MAX team up to keep Periscope out of the lower 640k...so you can have both quality memory management and a resident debugger.

If you need a debugger but have no room left in the lower 640K of your 386™ or 486™ system, Periscope now has a new SOFTWARE ONLY solution for you — Periscope/EM.

- Periscope/EM takes advantage of the extended memory you already have, rather than using any memory in the lower 640K.
- No runaway program can corrupt Periscope/EM because it's write-protected.
- You don't have to use a slot since there's no board to install.
- You can add an optional break-out switch to recover from crashes.
- Periscope/EM costs only $295, half the cost of Periscope Model I, with most of the same functionality.

Periscope/EM requires 386MAX® or BlueMAX™, version 5.11 or later, a 386 or 486 system with about 300K of extended memory, 32K of memory between 640K and one megabyte, DOS 3.0 or later.

New Debugging Tools

"I needed a means to debug interrupt handlers where I could really see what was going on. The hardware trace buffer is great. I was able to debug code in 3 days that I have been trying to debug for months!"

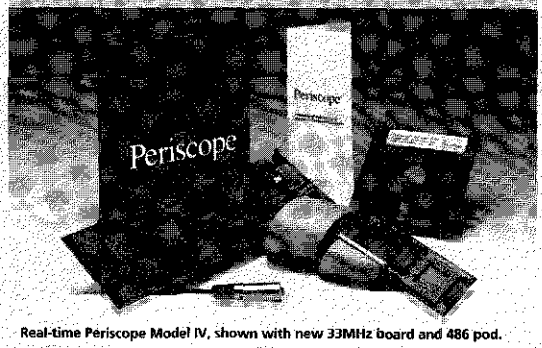
...writes Peg Sestrich with Prime Computer, on why she chose Periscope Model IV.

If you need more than your current software-based debugger gives you, Periscope Model IV with these advanced NEW features, may be just what you're looking for.

- Supports 386 and 486 systems running up to 33MHz with in-ICE-like capabilities.
- Real-time hardware trace buffer holds up to 16K CPU events.
- Enhanced analysis and display of trace buffer makes you more productive.
- Periscope/EM functionality, built-in at no extra cost, saves you money.

Just call toll-free 800/722-7006 for details or to order.

The Periscope Company, Inc.
1197 PEACHTREE ST
PLAZA LEVEL
ATLANTA, GA 30361
404-875-8080
FAX 404-872-1973



Real-time Periscope Model IV, shown with new 33MHz board and 486 pod.

Circle 17 on Reader Service Card

Imagine...

Information on Demand

Imagine having the information you want at your fingertips. Imagine that it's free.

Programmer's Journal Reader Service Cards — where imagination and reality come together.

Valuable information on products and services can be yours for **FREE** — we even pay the postage! Just tear out the ***PJ*** Reader Service Card opposite the Ad Index on p. 80. Circle the numbers from the ads or editorial mentions of the items you want to know more about.

Now, imagine yourself filling out the card and sending it in . . . today.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
t_esi           dd      ?
t_edi           dd      ?
t_es            dw      2 dup(?)
t_cs            dw      2 dup(?)
t_ss            dw      2 dup(?)
t_ds            dw      2 dup(?)
t_fs            dw      2 dup(?)
t_gs            dw      2 dup(?)
t_ldt           dw      2 dup(?)
t_tbit          dw      ?
t_iobase        dw      ?
t_iomap         db      IO_MAP_SIZE     dup(?)
t_iopad         dw      0ffh            ;follow last map byte with 0ffh
tss_dat         ends
```

LISTING 2

```
;iodat.inc - data for I/O Monitor
;
iostack3        segment para stack 'STACK' use16
io_stack3       dw      256 dup('S')
io_sp3          label   word
iostack3        ends

iostack         segment para public 'STACK' use16
STACK_BEG       EQU     $
io_stack        dw      256 dup('S')
io_sp           label   word
STACK_END       EQU     $
iostack         ends

gdt_seg         segment para public 'data16' use16
                public  sel_databs
                public  sel_stack
                public  sel_video
                public  sel_data
                public  sel_tss_alias

GDT_BEG         EQU     $
sel_null        seg_descrip     <>
sel_tss         seg_descrip     <>
sel_tss_alias   seg_descrip     <>
sel_gdt_alias   seg_descrip     <>
sel_isrcode     seg_descrip     <>
sel_code        seg_descrip     <>
sel_databs      seg_descrip     <>
sel_stack       seg_descrip     <>
sel_video       seg_descrip     <>
sel_data        seg_descrip     <>
GDT_END         EQU     $
gdt_seg         ends

idt_seg         segment para public 'data16' use16
IDT_BEG         EQU     $
                DEFIDTI 0
idt     <isrcode:int1_isr,gdt_seg:sel_isrcode,,PRESENT+(DPL0 shl 5)+INT_GATE,>
                DEFIDTI 2
                idt     <,,,PRESENT + (DPL0 shl 5) + INT_GATE,>

                irp     z, <4, 5, 6, 7>
                DEFIDTI &z
                endm

                irp     z, <8, 9, 0ah, 0bh, 0ch, 0dh, 0eh, 0fh>
                DEFIDTE &z
                endm

 idt        (1fh - 0fh)  dup(<,,,PRESENT + (DPL0 shl 5) + INT_GATE,>)

                irp     z, <20h, 21h, 22h, 23h, 24h, 25h, 26h, 27h>
                DEFIDTI &z
                endm
 idt        (6Fh - 27h)  dup(<,,,PRESENT + (DPL0 shl 5) + INT_GATE,>)

                irp     z, <70h, 71h, 72h, 73h, 74h, 75h, 76h, 77h>
                DEFIDTI &z
                endm

IDT_END         EQU     $
idt_seg         ends

data            segment para public 'data16' use16
                public  start_port
                public  end_port
                public  old_int1
                public  old_user_int
                public  io_table
                public  IO_TAB_ENTRIES
                public  access_port
                public  access_info
                public  store_buf
                public  store_buf_end
                public  store_ptr
                public  display_loc
                public  trap_status

DATA_BEG        EQU     $
bad_2nd_msg     db      '2nd port number must be >= first port.', BEEP, CR,
LF, '$'
bad_arg_msg     db      'Invalid port address specified.', BEEP, CR, LF, '$'
start_port      dw      0
end_port        dw      0
old_int1        dd      ?
old_user_int    dd      ?
access_port     dw      ?
access_info     db      ?
trap_status     db      0

io_table        io_struc        <0e4h, INPUT+CONSTANT+ABYTE>
                io_struc        <0e5h, INPUT+CONSTANT+AWORD>
                io_struc        <0e6h, OUTPUT+CONSTANT+ABYTE>
                io_struc        <0e7h, OUTPUT+CONSTANT+AWORD>
                io_struc        <0ech, INPUT+ABYTE>
                io_struc        <0edh, INPUT+AWORD>
                io_struc        <0eeh, OUTPUT+ABYTE>
                io_struc        <0efh, OUTPUT+AWORD>
IO_TAB_ENTRIES  EQU     ($ - io_table)/size io_struc

store_buf       db      BUF_SIZE dup(0)
store_buf_end   equ     $
store_ptr       dw      offset store_buf

gdtl            EQU     THIS FWORD
gdtlim          dw      (GDT_END-GDT_BEG)-1
gdtadrs         dd      ?

idtl            EQU     THIS FWORD
idtlim          dw      (IDT_END-IDT_BEG)-1
idtadrs         dd      ?

display_loc     dw      0               ;offset into video page
DATA_END        EQU     $
data            ends

tss_seg         segment para public 'data16' use16
TSS_BEG         EQU     $
                tss_dat <>
TSS_END         EQU     $
tss_seg         ends
```

LISTING 3

```
;ioinit (main) and ioisr modules for I/O Monitor merged for convenience
;Copyright 1991 ASMicro Co. 5/19/91  Rick Knoblaugh

                .386P
                include iomon.inc
                include iodat.inc
isrcode         segment para public 'icode16' use16   ; must define first
isrcode         ends
code            segment para public 'code16' use16
                assume cs:code, ds:data, es:data

start           proc    far
                push    ds              ;save psp seg
                mov     ax, data
                mov     es, ax
                call    get_args        ;get io ports
                mov     ax, data
                mov     ds, ax
                jnc     start_200       ;continue if args  ok
start_100:      mov     ah, DOS_PRT_STRING
                int     21h
                mov     ax, 4c01h
                int     21h
start_200:      call    setup_ints      ;take over user int

                call    init_gdt
                call    init_tss
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                mov     ax, data
                mov     ds, ax
                assume  ds:data

                cli                     ;no ints until protected mode

                mov     ax, gdt_seg
                movzx   eax, ax
                shl     eax,  4
                mov     gdtadrs, eax

                mov     ax, idt_seg
                movzx   eax, ax
                shl     eax,  4
                mov     idtadrs, eax

                call    reprogram_pic

                lgdt    gdtl
                lidt    idtl

                mov     dx, iostack3    ;get stack segment
                mov     bx, sp          ;and pointer
                mov     eax, cr0
                or      eax, 1          ;turn on protected mode bit
                mov     cr0, eax        ;go into protected mode

;jump to clear prefetch queue
                db      0eah            ;far jump
                dw      offset code:start_400
                dw      gdt_seg:sel_code
start_400:      mov     ax, offset gdt_seg:sel_tss
                ltr     ax
                xor     ax, ax
                lldt    ax              ;null ldt
                mov     ax, seg data
                movzx   eax, ax
                push    eax             ;null gs
                push    eax             ;null fs
                push    eax             ;null ds
                push    eax             ;null es

                push    0
                push    dx              ;stack segment
                push    0
                push    bx              ;stack pointer

                push    2               ;VM bit set in upper eflags
                push    3000h           ;NT=0, IOPL=3, CLI in lower eflags
                push    0
                push    seg code        ;cs of where to return
                push    0
                push    offset code:start_500  ;ip of where to return

;Must ensure that Nested Task bit is not set in eflags.  If it were,
;processor would attempt to switch to a task via the selector in
;the TSS backlink field.  Since that field is now zero, an invalid TSS
;fault would occur.
                pushf
                pop     ax
                and     ax, NOT NT_FLAG
                push    ax
                popf
                iretd
start_500:      pop     bx                  ;begin vm86 task here
                sti                         ;get saved psp seg
                mov     dx, code + 1        ;interrupts ok now
                sub     dx, bx              ;init code we are dropping
                mov     al, OK              ;exit code
                mov     ah, DOS_TSR_FUNC
                int     21h
start           endp

reprogram_pic   proc    near
                in      al, 21h
                mov     ah, al
                mov     al, 11h         ;init
                out     20h, al
                mov     al, 20h         ;irq0 to int 20h
                out     21h, al
                jmp     short $ + 2
                jmp     short $ + 2
                mov     al, 4
                out     21h, al
                jmp     short $ + 2
```

continued

Programmer Power Tools

Introducing: BASH™
A High Performance Key-Indexed Record Management System for DOS

The power of a database without the hassles! Use the BASH tool kit to quickly develop complex, high performance data management applications. BASH functions allow your application to quickly retrieve, insert, update, and delete records. Access records sequentially or randomly using single or multiple keys. You may choose the widely acclaimed B-tree indexing method, a hashed indexing method or both. Record locking is provided for multitasking and LAN applications. The highly optimized BASH kernel uses only 25K and is accessible from most DOS programming languages. BASH really is the best choice when it comes to developing data management applications.

$89 (object code only)
$249 (C source code)

DOS Multitasking with MultiDos Plus™

Elegant, easy to use, powerful DOS multitasking. Discover the product that OEMs around the world use for their critical applications. Thousands of customers and five years of proven reliability make MultiDos Plus the best choice for your multitasking platform.

- *Fast task context switch time*
- *Very flexible prioritized preemptive task scheduling*
- *Small memory overhead (64K)*
- *Complete complement of API functions at no extra charge*
- *Task execution in LIM 4.0 EMS lets your application grow without leaving DOS*
- *Excellent for real-time turnkey systems*
- *In-depth documentation with lots of sample software*

$99 (object code only)

Call today for complete information, demo or to order.
(800) 678-2141
(508) 651-0091
FAX (508) 655-8860
BBS (508) 650-9552

Nanosoft Inc.
13 Westfield Road
Natick, MA 01760

VISA MasterCard

Circle 18 on Reader Service Card

The Programmer's Corner
— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today: (301) 596-1180
(301) 995-3744
(301) 621-4636
300-9600 Baud
24 Hours-A-Day

Only $50 for a one year subscription

— TPC uses FastComm 9600 V.32 Modems —

TPC

The Programmer's Corner
5430 F Lynx Lane, Suite 224
Columbia, MD 21044

Circle 19 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                jmp     short $ + 2
                mov     al, 1
                out     21h, al
                jmp     short $ + 2
                jmp     short $ + 2
                mov     al, ah
                out     21h, al
                ret
reprogram_pic   endp

;get_args - retrieve starting and ending i/o addresses from
;           the command line.
;           Enter:  ds=PSP seg
;           Exit:  If invalid, carry set and dx=offset msg
get_args        proc    near
                cld
                xor     si, si
                sub     ch, ch
                mov     cl, [si].arg_len        ;get cmd line length
                lea     si, [si].arg_off        ;and offset
                call    skip_white_sp
                mov     dx, offset es:bad_arg_msg
                jcxz    get_a800                ;if no args
                call    get_validate            ;get first i/o port
                jc      get_a999
                mov     es:start_port, bx       ;save it
                call    skip_white_sp
                jcxz    get_a800                ;if no 2nd arg
                call    get_validate            ;get 2nd i/o port
                jc      get_a999
                mov     es:end_port, bx         ;save it
                cmp     bx, es:start_port       ;2nd arg >= 1st arg
                jae     get_a900
                mov     dx, offset es:bad_2nd_msg
get_a800:       stc
                jmp     short get_a999
get_a900:       clc
get_a999:       ret
get_args        endp

;get_validate - Retrieve and validate an ASCII hex number.
;               Enter: si ptr to data
;                      cx max len
;               Exit: data at si now binary
;                     cx decremented by length of number
;                     If invalid, carry set and dx=&err_msg
;                     bx=binary value of number
get_validate    proc    near
                xor     bx, bx
get_v100:       cmp     byte ptr [si + bx], ' '
                je      get_v200
                cmp     byte ptr [si + bx], CR
                je      get_v200
                inc     bx
                jmp     short get_v100
get_v200:       sub     cx, bx                  ;cmd line chars left
                push    cx
                mov     cx, bx
                xor     bx, bx
get_v300:       sub     ah, ah
                lodsb                           ;get a digit
                call    asc2bin                 ;validate and convert
                jc      get_v900                ;exit if invalid
                push    cx
                jcxz    get_v500
                dec     cx                      ;position from right - 1
                shl     cl, 2
                shl     ax, cl                  ;get nibble into position
get_v500:       or      bx, ax                  ;build number
                pop     cx
                loop    get_v300
                clc                             ;ok
get_v900:       pop     cx
                ret
get_validate    endp

asc2bin         proc    near
                cmp     al,  'a'
                jb      asc2b_100
                and     al, 0dfh                ;force to upper case
asc2b_100:      cmp     al, '0'
                jb      asc2b_800
                cmp     al, '9'                 ;see if 0-9 or A-F
                ja      asc2b_500
                sub     al, '0'                 ;convert to binary
                jmp     short asc2b_600

asc2b_500:      cmp     al, 'F'
                ja      asc2b_800
                sub     al, 'A' - 10
asc2b_600:      clc
                jmp     short asc2b_900
asc2b_800:      mov     dx, offset es:bad_arg_msg
                stc
asc2b_900:      ret
asc2bin         endp

skip_white_sp   proc    near
                cmp     byte ptr [si], ' '
                je      skip_w200
                cmp     byte ptr [si], TAB
                je      skip_w200
                jmp     short skip_w900
skip_w200:      inc     si
                loop    skip_white_sp
skip_w900:      ret
skip_white_sp   endp

setup_ints      proc    near
                mov     bx, USER_INT
                mov     di, offset old_user_int
                mov     cx, isrcode
                mov     dx, offset isrcode:user_int_isr
                call    get_int
                ret
setup_ints      endp

;get_int - For a given interrupt vector, store contents and
;          load with new isr address.
;          bx = int number
;          es:di = location to store contents
;          dx = offset new isr
;          cx = cs of new isr
get_int         proc    near
                cld
                push    ds
                xor     ax, ax
                mov     ds, ax
                shl     bx, 2
                mov     ax, [bx]
                stosw
                mov     ax, [bx].d_segment
                stosw
                cli
                mov     [bx].d_offset, dx
                mov     [bx].d_segment, cx
                sti
                pop     ds
                ret
get_int         endp

init_gdt        proc    near
                mov     ax, gdt_seg
                mov     ds, ax
                assume  ds:gdt_seg

                mov     dx, tss_seg
                movzx   edx, dx                 ;base data segment
                mov     ecx, (TSS_END - TSS_BEG ) - 1  ;limit
                mov     ah, TSS_DESC
                mov     si, offset sel_tss
                call    make_entry

                mov     dx, tss_seg
                movzx   edx, dx                 ;base data segment
                mov     ecx, (TSS_END - TSS_BEG ) - 1  ;limit
                mov     ah, RW_DATA             ;alias as r/w for editing
                mov     si, offset sel_tss_alias
                call    make_entry

                mov     dx, gdt_seg
                movzx   edx, dx                 ;base data segment
                mov     ecx, (GDT_END - GDT_BEG ) - 1  ;limit
                mov     ah, RW_DATA             ;alias as r/w for editing
                mov     si, offset sel_gdt_alias
                call    make_entry

                mov     dx, isrcode
                movzx   edx, dx                 ;base of isr code segment
                mov     ecx, 0ffffh             ;max segment size
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                mov     ah, ER_CODE
                mov     si, offset sel_isrcode
                call    make_entry

                mov     dx, code
                movzx   edx, dx                 ;base code segment
                mov     ecx, 0ffffh             ;max segment size
                mov     ah, ER_CODE
                mov     si, offset sel_code
                call    make_entry

                xor     edx, edx                ;zero base
                mov     ecx, 0fffffh            ;base memory size
                mov     ah, RW_DATA
                mov     si, offset sel_databs
                call    make_entry

                mov     dx, iostack
                movzx   edx, dx                 ;base stack segment
                mov     ecx, (STACK_END - STACK_BEG  ) - 1  ;limit
                mov     ah, RW_DATA
                mov     si, offset sel_stack
                call    make_entry

                mov     dx, data
                movzx   edx, dx                 ;base data segment
                mov     ecx, (DATA_END - DATA_BEG  ) - 1  ;limit
                mov     ah, RW_DATA
                mov     si, offset sel_data
                call    make_entry

                int     11h                     ;equipment check
                mov     edx, 0b800h             ;color segment
                and     al, 30h                 ;monitor bits
                cmp     al, 30h                 ;30h=monochrome
                jne     short init_gdt500
                mov     edx, 0b000h             ;monochrome segment
init_gdt500:    mov     ecx, VID_PAGE_SIZE - 1  ;page size - 1
                mov     ah, RW_DATA
                mov     si, offset sel_video
                call    make_entry
                ret
init_gdt        endp

;make_entry - Load a GDT entry from information passed as follows:
;             ds=gdt segment
;             si=offset of gdt entry to load
;             ah=type | dpl
;            ecx=limit
;            edx=base segment (convert it to linear)
;            Always set for byte granularity and 16 bit size
make_entry      proc    near
                shl     edx, 4                  ;convert seg to linear
                mov     [si].seg_limit_low, cx
                mov     [si].seg_base_low, dx
                shr     edx, 16
                mov     [si].seg_base_mid, dl
                mov     [si].seg_type_dpl, ah
                shr     ecx, 16
                and     cl, 0fh                 ;limit and byte gran
                mov     [si].seg_limit_gran, cl
                mov     [si].seg_base_top, dh
                ret
make_entry      endp

;init_tss - Initilize TSS with base of I/O bit map and set
;          appropriate bits in I/O bit map per cmd line.
init_tss        proc    near
                mov     ax, tss_seg
                mov     ds, ax
                assume  ds:tss_seg
                xor     si, si

                mov     ax, offset gdt_seg:sel_stack
                mov     [si].t_ess0, ax
                mov     ax, offset iostack:io_sp
                movzx   eax, ax
                mov     [si].t_esp0, eax

                lea     bx, [si].t_iomap
                mov     [si].t_iobase, bx

                mov     cx, es:start_port       ;start of range to monitor
                mov     dx, cx
                and     cl, 7                   ;get non byte boundary
                mov     al, 1                   ;first bit position
                shl     al, cl                  ;get out corresponding bit
                shr     dx, 3                   ;start_port/8
                mov     cx, es:end_port
                sub     cx, es:start_port
                inc     cx
                add     bx, dx                  ;starting offset in map
init_t100:      or byte ptr [bx], al            ;turn on permission bit
                rcl     al, 1                   ;next bit position
                jnc     init_t300
                inc     bx
                rcl     al, 1
init_t300:      loop    init_t100
                ret
init_tss        endp
code ends

isrcode         segment para public 'code16' use16
                assume cs:isrcode, ds:nothing, es:nothing

;Interrupt Service Routines for I/O Monitor

        irp     z, <0, 2, 3, 4, 5, 6, 7>
        DOINT   &z
        endm

        irp     z, <8, 9, 0ah, 0bh, 0ch, 0dh, 0eh, 0fh>
        DOEXCP  &z
        endm

        irp     z, <20h, 21h, 22h, 23h, 24h, 25h, 26h, 27h>
        DOEXCPH &z
        endm

        irp     z, <70h, 71h, 72h, 73h, 74h, 75h, 76h, 77h>
        DOINT   &z
        endm

;int1_isr -  ISR for single step interrupt.  If trap_status
;            has EXPECT_INT1 bit set, this is the trap
;            that occurs just after the I/O instruction
;            we are monitoring executes.  Store the I/O data
;            from ax into our buffer.
int1_isr        proc    far
                push    bp
                mov     bp, sp
                push    bx
                push    ds
                mov     bx, offset gdt_seg:sel_data
                mov     ds, bx
                assume  ds:data
                test    trap_status, EXPECT_INT1 ;expecting an int 1?
                jnz     int1_050
                pop     ds
                pop     bx
                push    1
                jmp     pass_thru               ;if not, do old int 1
 int1_050:      push    ax
                push    cx
                push    dx
                mov     trap_status, 0
                and     [bp].s_eflags, NOT TRAP_FLAG ;no more single step
                mov     bx, store_ptr
                mov     cl, access_info
                or      cl, CONTAINS_DATA       ;indicate data stored
                mov     [bx].buf_info, cl       ;store info re i/o
                mov     dx, access_port
                mov     [bx].buf_port, dx       ;store port number
                test    cl, AWORD               ;word access?
                jnz     int1_100
                sub     ah, ah                  ;if not, clear half
int1_100:       mov     [bx].buf_data, ax
                add     bx, size buf_record     ;advance buf ptr
                cmp     bx, store_buf_end - size buf_record  ;at end?
                jb      int1_200
                mov     bx, offset store_buf
int1_200:       mov     store_ptr, bx           ;save new ptr
                and     cx, (AWORD OR ABYTE)    ;number of bits
                mov     ah, 1                   ;indicate set
                mov     dx, access_port
                call    do_bit_map      ;set the bits again
                pop     dx
                pop     cx
                pop     ax
                pop     ds
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                pop     bx
                pop     bp
                iretd
int1_isr        endp

;except_handler - Process as follows:
;                       Int 0dh - Go look for software int or
;                                 I/O instruction.
;                       Any other
;                       exception - go display exception number
;                                   and halt.
except_handler  proc    near
                mov     bp, sp
                cmp     [bp].e_pushed_int , GEN_PROT_EXCEP
                je      gen_prot_isr
                mov     ax, [bp].e_pushed_int ;int in ax, go display
                jmp short fatal_error
except_handler  endp

fatal_error     proc    near
                call    display_it
                jmp     $
fatal_error     endp

;display_it - Display hex number on screen at next display offset
;            Enter:  number in AX processor in protected mode
;            All registers saved
display_it      proc    near
                pusha
                push    ds
                push    es

                mov     dx, offset gdt_seg:sel_data
                mov     ds, dx                  ;get our data segment
                assume  ds:data
                mov     dx, offset gdt_seg:sel_video
                mov     es, dx                  ;and video segment
                mov     di, display_loc
                xchg    al, ah                  ;print MSB first
                call    put_hex_digit
                xchg    al, ah                  ;get LSB
                call    put_hex_digit
                add     di, 2                   ;past space and attribute
                cmp     di, VID_PAGE_SIZE
                jb      display_i100
                xor     di, di                  ;back to start of page
 display_i100:  mov     display_loc, di
                pop     es
                pop     ds
                popa
                ret
display_it      endp

put_hex_digit   proc    near
                push    ax
                mov     cx, 2                   ;2 digits in al
                mov     ah, al
put_hex_100:    shr     al, 4
                cmp     al, 9
                ja      put_hex_200
                add     al, '0'
                jmp     short put_hex_300
put_hex_200:    add     al, 'A' - 10
put_hex_300:    cld
                stosb
                inc     di                      ;past attrib
                mov     al, ah
                shl     al, 4
                loop    put_hex_100
                pop     ax
                ret
put_hex_digit   endp

;pass_thru - This procedure is JMPed to by any interrupt handler that
;            wishes to pass control to the original ISR per the
;            interrupt vector table.
;            Entry:
;                 See stack_area struc for stack layout.
;                 Any error code has been removed from stack.
;                 EIP on stack has been adjusted if
;                 necessary.
pass_thru       proc    near
                mov     bp, sp
                pushad
                push    ds
                mov     ax, offset gdt_seg:sel_databs
                mov     ds, ax                  ;address all base memory
                movzx   ebx, [bp].s_ss          ;user stack
                shl     ebx, 4                  ;make linear
                mov     edx, [bp].s_esp         ;user stack pointer
                sub     edx, 6                  ;flags, cs, ip
                mov     [bp].s_esp, edx         ;adjust it
                mov     eax, [bp].s_eflags      ;put on flags
                mov     [ebx][edx].user_flags, ax

;change flags on stack so that original ISR will be entered with
;interrupts cleared and trap flag cleared to be consistent with their
;state upon entering an ISR (the normal way).
                and     ax, not (TRAP_FLAG + INT_FLAG)
                mov     [bp].s_eflags, eax      ;put back flags
                mov     ax, [bp].s_cs           ;put on user cs
                mov     [ebx][edx].user_cs, ax
                mov     eax, [bp].s_eip         ;put on ip
                mov     [ebx][edx].user_ip, ax
                movzx   ebx, [bp].s_pushed_int  ;get int number
                movzx   eax, [ebx * 4].d_offset ;offset portion
                mov     [bp].s_eip, eax
                mov     ax, [ebx * 4].d_segment ;segment portion
                mov     [bp].s_cs, ax
                pop     ds
                popad
                add     sp, 2                   ;get rid of int number
                pop     bp
                iretd
pass_thru       endp

;gen_prot_isr - JMP here if int 0dh.  Process as follows:
;               Look for software int.  If found, go route to
;               appropriate ISR.
;
;               Look for I/O instructions we currently
;               support.  If found, store port, size and
;               direction of I/O.  Also, set trap flag and
;               return (we will get control at int 1 to
;               inspect data).
;
;               If other than software int or I/O, go
;               display 0dh and halt.
gen_prot_isr    proc    near
                push    ds
                pushad
                mov     bx, offset gdt_seg:sel_databs
                mov     ds, bx
                movzx   ebx, [bp].e_cs  ;get cs of user instruction
                shl     ebx, 4          ;make linear
                add     ebx, [bp].e_eip ;add ip
                mov     ax, [ebx]       ;get bytes at cs:ip
                cmp     al, INT_OPCODE
                jne     get_prot100
                inc     [bp].e_eip      ;get past the 0cdh
get_prot050:
                inc     [bp].e_eip

;Adjust stack so that error code goes away and int number retrieved from
;instruction goes in spot on stack where pushed int number is (for stacks
;with no error code).  Stack will be the way pass_thru routine likes it.
                mov     bx, [bp].e_pushed_bp
                shl     ebx, 16         ;get into high word
                mov     bl, ah          ;interrupt number
                mov     [bp].e_errcode, ebx
                popad
                pop     ds
                add     sp, 4           ;error code gone
                jmp     pass_thru
get_prot100:    cmp     al, INT3_OPCODE
                jne     get_prot150
                mov     ah, 3           ;interrupt 3
                jmp     short get_prot050
get_prot150:    mov     bx, offset gdt_seg:sel_data
                mov     ds, bx
                mov     bx, offset io_table
                mov     cx, IO_TAB_ENTRIES
get_prot200:    cmp     al, [bx].io_opcode
                jne     get_prot300
                mov     trap_status, EXPECT_INT1
                mov     cl, [bx].io_info ;get info about instruction
                mov     access_info, cl
                mov     access_port, dx  ;save port
                test    cl, CONSTANT     ;is port number in instruction?
                jz      get_prot250      ;if not, we have it
```

continued

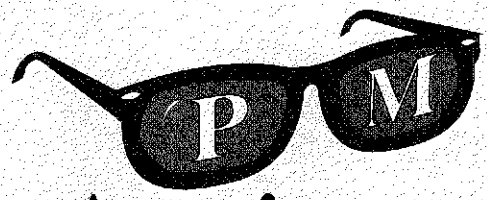
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                xchg    ah, al          ;ah = 2nd byte of instruction
                sub     ah, ah
                mov     access_port, ax ;save port
                mov     dx, ax
get_prot250:    and     cx, (AWORD OR ABYTE)   ;number of bits
                sub     ah, ah          ;indicate clear
                call    do_bit_map

                popad
                pop     ds
                or      [bp].e_eflags, TRAP_FLAG ;single step i/o

                add     sp, 2           ;int number pushed
                pop     bp
                add     sp, 4           ;error code
                iretd
get_prot300:    add     bx, size io_struc ;advance to next table entry
                loop    get_prot200
                mov     ax, [bp].e_cs   ;get cs of user instruction
                call    display_it
                mov     eax, [bp].e_eip ;add ip
                call    display_it
                popad
                pop     ds
                mov     ax, [bp].e_pushed_int
                jmp     fatal_error
gen_prot_isr    endp

;do_bit_map - For the number of ports specified, clear/set
;             corresponding I/O permission map bits.
;             Enter:  Ah = 0 clear, ah = 1 set
;                     dx = starting port
;                     cx = number of ports
;  All registers saved.

do_bit_map      proc    near
                push    ax
                push    bx
                push    cx
                push    dx
                push    ds
                mov     bx, offset gdt_seg:sel_tss_alias
                mov     ds, bx
                assume  ds:tss_seg
                mov     bx, t_iomap
                push    cx
                mov     cx, dx                  ;port
                and     cl, 7                   ;get non byte boundary
                mov     al, 1                   ;first bit position
                shl     al, cl                  ;get out corresponding bit
                shr     dx, 3                   ;start_port/8
                pop     cx
                add     bx, dx                  ;starting offset in map
do_bit100:      or      ah, ah
                jnz     do_bit200
                mov     dl, al
                not     dl
                and     byte ptr [bx], dl       ;turn off permission bit
                jmp     short do_bit250
do_bit200:      or byte ptr [bx], al            ;turn on permission bit
do_bit250:      rcl     al, 1                   ;next bit position
                jnc     do_bit300
                inc     bx
                rcl     al, 1
do_bit300:      loop    do_bit100
                pop     ds
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
do_bit_map      endp


;user_int_isr - return buffer address to caller.
;             Exit: dx:bx= far ptr to store_buf

user_int_isr    proc    near
                mov     dx, data
                mov     bx, offset data:store_buf
                iret
user_int_isr    endp
isrcode         ends

                end start
```


Conquer OS/2 !!

Now OS/2 can respond to your every whim

Introducing:



Assistant

PM Assistant is the **first** and **only** Automation and Customization Utility for OS/2 Presentation Manager

- Automate repetitive tasks with **mouse and keyboard input macros.**
- Play keystrokes to any PM application or full-screen session.
- Start, position, and initialize your PM applications **automatically.**
- Switch to and position PM applications and full-screen sessions with user-defined **hot-keys.**
- Execute any macro from within any PM application or full-screen session with user-defined **hot-keys.**

- Set, view, and change *any* program's **OS/2 process priority.**
- Assign **accelerator keys** and rearrange the menu commands of *any* PM application.
- Speed dialog box interaction with Speed Button.
- Prevent damage to your monitor with **Screen Saver.**
- Protect your computer from unauthorized access with **Password Protection.**
- Create turn-key procedures or demonstrations of your PM programs with the **demonstration utility.**

Now, all your favorite tools in Dos and Windows are available for Presentation Manager along with features unique to OS/2. PM Assistant comes with on-line help, an unconditional satisfaction guarantee, and a **free program** - DiskWatch, a disk free-space monitor. PM Assistant runs on OS/2 versions 1.10 or greater, and is now available at a special introductory price:

PM Assistant.....$125.00

Take control of OS/2, order today!

1-800-745-7757

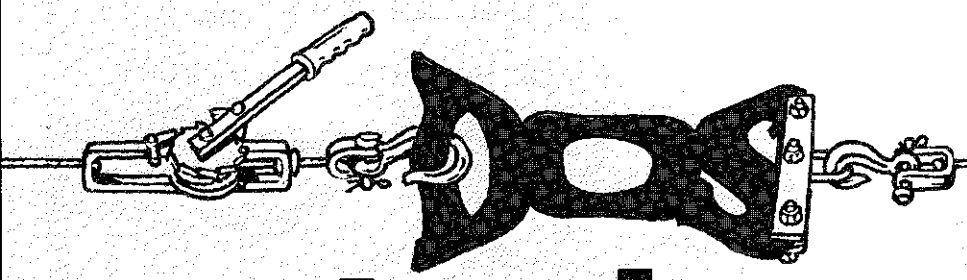
Utilis Advanced Tools Now

Trademarks are property of their respective holders.

P.O. Box 15905
Seattle, WA 98115
(206) 467-4025

Circle 21 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



extenders

Stretching the limits

By M. Steven Baker

DOS has incredible staying power. Although it won't be long before Intel releases 80586 chips, MS-DOS—the dominant operating system on the *x*86 chips—has seen only modest changes since version 2.0. DOS 3 added 16-bit FATs and network support, but versions 3.0 and 3.2 were terribly bug-ridden. Compaq DOS 3.31 and DOS 4.*x* broke the 32MB disk-partition barrier—a feat that third-party software such as SpeedStor, Disk Manager, Vopt, and Compaq's EXTDISK.SYS had accomplished some time earlier.

DOS 5 is a welcome improvement over its predecessors, even though many of the features it adds should have been available since the introduction of Windows 2.1 and HIMEM.SYS. The PC world has benefited from the feature in the 80286 chip that allowed an extra 64K (minus 16 bytes) of memory to be accessible when segment address FFF0 was used. DOS 5 exploits this bug/feature to load much of DOS into that magic-memory land; of course, Windows 2 used the same technique when using HIMEM.SYS way back when. At long last, DOS 5 sports a decent visual shell for those who dislike the DOS prompt. This shell represents a trade-off between user interface and memory for DOS applications. On a 386, a modest task switcher has been added. Some valuable utilities (the UNDELETE and UNFORMAT commands from Central Point Software) are also incorporated, but DOS 5 brings little new to the overall basic operating system.

STRETCHING THE LIMITS

The IBM AT was introduced in 1984, sporting the spiffy new 80286. In 1986, the code phrase for the new protected mode operating system being developed by Microsoft and IBM for the 80286 was DOS 5. The skeptics who believed in DOS and its rich set of user applications were the first to develop DOS extenders.

Back in 1987, as an OS/2 disbeliever, I predicted that DOS extenders would likely have at least a five-year life span. The five years will soon be up, and the prospects for their continued existence seem brighter than ever. One surprising fact is that the four principal players at the time—Rational Systems, Ergo Computing (formerly A.I. Architects), Phar Lap Software, and IGC—have remained the same. Microsoft has now joined the club with the DOS extender it built into Windows 3 (standard and enhanced modes), as has Salford Software (from Great Britain), which bundles a DOS extender with its Fortran (and soon other language) compilers. These are marketed in the United States by OTG Systems. MetaWare, the first compiler vendor to support the 386 protected mode under DOS, still plays a major part in this market.

Some things may persist, but much has changed since the first versions of these products. The early 286 DOS extenders were able to use the existing 16-bit DOS compilers by supplying patches to their runtime libraries for the protected mode, but 386 compilers were limited. Now we have an abundance of tools for 386 development. Most DOS compilers and development tools come bundled with a DOS extender to free them from DOS's 640K memory limits. Even Microsoft has quietly slipped in its 286 DOS extender from Windows 3 (DOSX.EXE) with this summer's update of the C6.0a compiler. DOS 5 may add back 30 to 40K for an application, but DOS extenders add megabytes.

IT'S ALWAYS SOMETHING

With the introduction of Windows, a new set of problems and opportunities has arisen for the vendors of DOS extenders. Similar scenarios have occurred in the past. In 1987, Phar Lap Software and Quarterdeck Systems joined forces to develop the Virtual Control Program Interface (VCPI). This headed off the potential problem of memory managers (such as DESQview's QEMM and Qualitas's 386MAX) and DOS extenders fighting for control of the 80386 in protected mode.

Except for IGC's XAM, VCPI support was swiftly added to all DOS extenders and memory managers, eliminating the possible conflict between them. (IGC's XAM was based on running DOS in the virtual-8086 mode, which made implementing VCPI much more difficult.) The VCPI standard (completed in 1989) is based on the memory manager being the server and the DOS extender being the client. Both programs run at ring 0, so there's no chance to virtualize devices and no way to protect one application from another. VCPI works because the DOS-extender software and the memory-manager software cooperate.

Since the development of VCPI,

M. Steven Baker is the editor of Programmer's Journal *and a co-author of* Extending DOS, *published by Addison Wesley.*

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS extenders and memory managers have responded to the XMS API for managing extended memory above 1MB. More recently, the Virtual DMA Services (VDS) specification has also been supported. This specification is needed for some hardware boards (such as SCSI adapters and hard-disk controllers) that incorporate their own DMA chips on the board. VDS ensures that programs (commonly, device drivers, tape backup software, etc.) can lock memory and retrieve an actual physical address that will be put on the system bus for DMA access. Without some sort of well-defined protocol, it's possible that a memory manager, DOS extender, or multitasker might page out the target physical memory block during a transfer and substitute another logical chunk of memory.

Now Windows 3 has added the DOS Protected-Mode Interface (DPMI). DPMI is designed to allow DOS extended applications to run under any 386 operating system or control program that supports it. The DPMI specification is available from Intel (see vendor information). The first product to be released with support for DPMI, Windows 3 acts as a DPMI host. DPMI hosts have since been promised for IBM's OS/2 2.0 and from a number of 386/Unix vendors.

For both 286 and 386 DOS extenders, the exciting changes are based on incorporating code to allow them to operate as DPMI clients. Programs that incorporate the new DOS-extender versions are currently able to run under Windows 3 in a DOS box and will run under future versions of OS/2 2 and Unix/386, as well as on plain old DOS. If such a range of options sounds like a new lease on life, it's because it is.

THE EMPEROR'S NEW CLOTHES

First, I should mention that implementing DPMI clients in DOS extenders causes some headaches for their vendors. These products must incorporate support for a number of different extended memory APIs to cope with all possible situations:

- Bottom-up extended memory allocation (first used by VDISK.SYS)
- Hook INT 15h (function 88h) interrupt handler for top-down memory allocation
- VCPI used by 386 memory managers (EMS and XMS simulators)
- XMS drivers (HIMEM.SYS)
- DPMI

Of these five methods, DPMI is clearly the most complex to incorporate. This is exacerbated by the fact that the DPMI 0.9 specification supported in Windows 3 is missing some of the functions required to provide all the features available under the DOS extenders when running stand-alone on plain DOS. Also, these products are all designed to work with either the 0.9 or 1.0 version of DPMI. As yet, there's no DPMI 1.0 host available with which to test a DPMI DOS-extender client, so the big question is whether the vendors will have to revise their DPMI client code after a real DPMI 1.0 server becomes available.

DOS 5 may add back 30 to 40K for an application, but DOS extenders add megabytes.

LOOK THROUGH ANY WINDOW

Before we look at the DOS-stretching products, let's discuss the big incentive for all this feverish activity—namely, Windows 3. Although it may not be as friendly or consistent as the Mac OS, Windows 3 is the hot user interface in DOS town. And after all the years of bad-mouthing DOS extenders, Microsoft has gone ahead and incorporated one into their new Windows runtime. The DOS extender in Windows even supports both 286 and 386 programs. You won't find much information about this in the Windows SDK manuals, so I'll point out a few places to find some. Although the manuals never directly mention DPMI services or DOS extenders, Windows 3 supplies a standard INT 21h DOS-stretching interface to applications. The *Windows SDK Reference—Volume 1* briefly documents the calls **DOS3Call** (eventually issues an INT 21h to DOS) and **NetBIOSCall** (issues the NETBIOS interrupt 5Ch).

Because only registers are used for entry and return values, both calls must be made from assembly language. These two routines are exported from KERNEL.EXE and are not defined in any Windows **include** files. The manual also states that using **DOS3Call** is faster than an INT 21h, presumably because you use a call gate instead of going through the interrupt descriptor table (IDT). Windows 3 can be run in either real, standard (286 protected), or enhanced (386 protected) mode. In the two protected modes, Windows acts as a DOS extender and transfers these DOS calls to the real-mode DOS system in the lower 640K of memory. Nowhere in the manuals will you find a list of the DOS function calls supported by this mechanism. Microsoft publishes a five-page technical note that describes the supported DOS calls and DPMI functions. Although undocumented, other DOS INT *x* and ROM BIOS calls are also supported from the protected modes. In general, most INT calls are passed on to real-mode DOS by the Windows kernel. For illustration, do an INT 10h call from a Windows program and you'll get garbage on your screen. The vendors of DOS extenders have the choice of using the Windows DOS extender or using DPMI to work around Windows.

A number of other DOS-like file I/O functions are mentioned in the *Windows SDK Reference*. These include **GetDriveType, GetTempDrive, GetTempFileName, _lclose, _lcreate, _lseek, _lopen, _lread, _lwrite, Openfile,** and **SetHandleCount.** The purposes of these functions all correspond reasonably well with their names, except for **OpenFile,** which can create, open, or delete a file.

Exploring the main Windows SDK library with a good librarian (such as SLR's OPTLIB) will turn up an assortment of undocumented selectors, data variables, and functions. A few of the more interesting ones are **EnableDOS, DisableDOS, EnableKernel, DisableKernel, GlobalMasterHandle, OpenPathname, DeletePathname, GetLastDiskchange, NoHookDOSCall, KbdRst, FileCdr, DoSignal,** and **GetCurPid.** You'll also find some undocumented exported functions that are related to selector handling. Some of

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

these are **GetTaskDS, PrestoChangoSelector, SelectorAccessRights, GetSelectorBase, SetSelectorBase, GetSelectorLimit, SetSelectorLimit,** and **AllocSelectortoArray.** Undocumented data selectors include __ROMBIOS, __0000H, __0040H (which doesn't point there), __A000H, __C000H, __D000H, __E000, __F000H, and _FFFE_FARFRAME. A slew of undocumented exported functions deal with spooling printed output and network functions. But the most intriguing of all are the undocumented functions **Death** and **Resurrection.**

STEPPING UP TO 32 BITS

Volume 2 of the *Windows SDK Reference* briefly discusses (in Appendix E) the WINMEM32.DLL, which can be used by Windows apps for 32-bit memory management. This DLL supplies a standard method for implementing a 32-bit flat memory model. The SDK documentation states that using these calls guarantees that your application will run with future Windows versions. WINMEM32 contains eight functions to enable accessing 32-bit memory from an application:

- **Global32Alloc** allocates a block of 32-bit memory
- **GlobalRealloc** changes the size of a 32-bit memory object
- **Global32Free** frees a 32-bit memory object
- **Global16PointerAlloc** converts a 32-bit pointer to a 16-bit far pointer alias
- **GlobalPointerFree** frees a 16-bit pointer alias
- **Global32CodeAlias** creates a code alias for a 32-bit memory object
- **Global32CodeAliasFree** frees a 32-bit code alias
- **GetWinMem32Version** returns the version of the DLL

The WINMEM32.DLL shipped with Windows SDK has a reputation for being buggy, and many developers have steered clear of it. For programmers, WINMEM32.DLL puts a pretty face on some DPMI calls. The original Windows 3 runtime had several bugs related to DPMI and real-mode programming, so Microsoft updated it in late 1990. If you have an old version of Windows (files dated 05-01-90), you should have it updated to version 3.0a.

DPMI is mentioned only in a short paragraph in a README file on one of the disks in the Windows Device Driver Kit, but it is part of Windows 3 in both standard and enhanced modes. One drawback to

LISTING 1

```
/* testdpmi.c - based on Intel 0.9 DPMI spec, use Microsoft C6 cl -AS */

#include <stdio.h>

unsigned short GoProtected();
unsigned short getdpmiInfo();

main()
{
    unsigned short regCS, regDS, regES, regSS, regSP;
    unsigned dpmiES, dpmiDI;
    unsigned short dpmiFlags, dpmiParas, dpmiCputype;

    struct {
        unsigned char minor;
        unsigned char major;
    }   dpmiVer;

    _asm mov    regCS,cs                ;store real mode CS
    _asm mov    regDS,ds                ; and DS for display

    printf("Operating in real mode with CS=%04xh DS=%04xh\n\n",
           regCS,regDS);

    if ( getdpmiInfo() ) {
        printf("A DPMI server was not found .., exiting\n");
        exit(1);
    }
    _asm {
        mov     dpmiES,es               ; also the DPMI entry point
        mov     dpmiDI,di               ; for later
        mov     dpmiVer,dx              ; Also the DPMI version
        mov     dpmiCputype,cl          ; the Processor type
        mov     dpmiFlags,bx            ; if BX bit 0=1, 32-bit apps ok
        mov     dpmiParas,si            ; Private data area required by
                                        ;   DOS extender in paragraphs
    }
    printf("DPMI version = %d.%d  entry point = %04x:%04xh\n",
                dpmiVer.major, dpmiVer.minor, dpmiES,dpmiDI);
    printf("Processor type is an 80%d86 and the dpmiFlags are %04xh\n",
        dpmiCputype, dpmiFlags);
    if (dpmiFlags && 1)
        printf("32-bit applications are supported\n");
    else
        printf("32-bit applications are not supported\n");

    printf("DPMI requires %04xh paragraphs of private data area\n\n",
           dpmiParas);

    if ( GoProtected() ) {
        printf("Not able to switch to protected mode or malloc failed\n");
        exit(1);
    }
    _asm {
        mov     regCS,cs                ;store registers we got from
        mov     regDS,ds                ; protected mode
        mov     regES,es
        mov     regSS,ss
        mov     regSP,sp
    }
    printf("We're in DPMI protected mode now with registers set as\n");
    printf("CS=%04xh  DS=%04xh ES=%04xh  SS=%04xh  SP=%04xh\n",
           regCS, regDS, regES, regSS, regSP);

    _asm mov    ah,4ch              ;the only documented way out
    _asm int    21h                 ; from DPMI is to use this DOS exit
}

unsigned short getdpmiInfo(void)
{
    _asm    mov ax,1687h
    _asm    int 2fh
}

unsigned short GoProtected(void)
{
    void far *dpmi_entry;

    _asm {
        mov     ax,1687h                ; just get the registers again
        int     2fh                     ;  it's easier
        or      ax,ax                   ; we should get a zero again
        jnz     dpmi_error

        mov     word ptr dpmi_entry,di  ;store the entry address
        mov     word ptr dpmi_entry+2,es
        or      si,si                   ;is memory needed ?
        jz      call_dpmi

        mov     bx,si                   ; allocate memory for DPMI
        mov     ah,48h                  ; private area
        int     21h
        jc      dpmi_error

    call_dpmi:
        mov     es,ax               ;pass segment of data area
        mov     ax,0                ;indicate we are a 16-bit app
        call    dpmi_entry          ;switch to protected mode
        xor     ax,ax               ;signal success
    }
    dpmi_error: ;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the Windows 3 implementation of DPMI is that it's available to both Windows and DOS applications in the enhanced mode, but available only to Windows applications in the standard (286) mode. In standard mode, Windows doesn't support a call for DOS apps to find the entry point of the DPMI host. So it's tough luck, folks.

Using the DOS extender supplied with Windows 3, it is now possible to write 286 and 386 apps that run under Windows and can access lots of memory. You may, however, need access to real-mode memory, interrupt vectors, or special services that are not openly provided by Windows: they may be there using DPMI, but it takes a good bit of work to make them function. Also, such applications will run *only* under Windows or some other DPMI host.

If you want to ship a single executable that will run under DOS or Windows, the DOS-extender vendors have the solution. Most of these products have been around in some form for four years. And over this time, their vendors have added a number of features that facilitate creating extended-DOS programs. Most of these products are also well supported by compilers and other development tools. Let's begin our tour with the new 286 DOS extenders.

PHAR LAP RIDES AGAIN

Earlier this year, Phar Lap software released the only truly new product in this lineup—the 286|DOS-Extender SDK ($495). Its name may not earn many points for creativity, but in other ways this product is a winning contender. It can execute as a DPMI client, so it is compatible with the DPMI server as implemented in Windows 3. The original DOS extenders (both 286 and 386) have supported the standard INT 21h DOS interface, but in protected mode. The Phar Lap 286|DOS-Extender breaks with tradition by supporting two programming interfaces: the DOS interrupt interface and the call-stack-based interface of OS/2 (VIO calls only—no Presentation Manager here). Although this sounds like more work than just supporting the DOS INT 21h mechanism, it's actually a much simpler approach. I'll outline why below.

The 286|DOS-extender is designed to work with Microsoft C versions 5.1, 6.0, and 6.0a. For the time being, that's it. If your preferred development environment is something else—such as Borland or Zortech C++, Turbo Pascal, a different C compiler (from Watcom, MetaWare, or JPI), or another language—well, this product as it now stands won't cut it for you.

In spite of these limitations, Phar Lap did derive a nice side benefit by having their runtime support the OS/2 character-mode API. They already have a windowing symbolic debugger available in Microsoft's CVP (OS/2 protected-mode CodeView debugger). RUN286 (the loader for 286 DOS-extender applications) can be used to execute most of the Microsoft C OS/2 (protected-mode) development tools under DOS on a 286 or 386 machine with extended memory. The Phar Lap tools also support DLLs, just as OS/2 does. In fact, Phar Lap ships a number of DLLs with the product, including support for named pipes, session manager calls, queues, and threads. They also include patch files for the Microsoft C OS/2 libraries for the three supported compiler versions. These add some functions that aren't available in OS/2, such as an **int86** function, and they also patch some other OS/2 routines. The library additions also provide access to the Phar Lap API (PHAPI.LIB), with support functions to ease the transition to the protected mode.

The WINMEM32.DLL shipped with Windows SDK has a reputation for being buggy, and many developers have steered clear of it.

The product comes with three perfect-bound manuals: the *Release Notes*, a *Developer's Guide*, and a *Reference Manual*. The *Developer's Guide* provides a number of examples (also included on disk) that illustrate features such as hooking interrupts, mixing real- and protected-mode code, and accessing DDE. The SDK can be used for development. An unlimited runtime license for bundling RUN286 with your application starts at $995 (some restrictions apply).

Under Windows 3 on a 386/486 (in enhanced mode only), 286|DOS-Extender acts as a DPMI client and supports virtual memory (or, more accurately, the Windows 3 enhanced mode supports virtual memory). When it's running stand-alone on a 286/386, only physical memory is available. Under Windows 3 in standard mode (as when running on a 286), an application is limited to less than the 640K that DOS provides. For more memory, an XMS driver (HIMEM.SYS) must be installed, and a PIF file must be created for your application that makes XMS memory available. Otherwise, the DOS extender is running in a limited DOS box. Under standard mode, 286|DOS-Extender does not offer virtual memory.

WINDOW MODES AND DPMI

Unlike DOS apps, Windows apps *do* have access to DPMI in the standard mode, allowing more memory space and machine control. The reason for this difference is clear enough: Windows does not provide access to the DPMI entry point from a DOS app in standard mode; to support DPMI in a DOS box would have reduced the size of the DOS box by 30 to 40K.

Listing 1 is a short program inspired by the 0.9 DPMI specification. It tests for the presence of DPMI, reports statistics on the DPMI host, switches into the protected mode, and then exits. You can test this code on machines running enhanced-mode Windows on a 386/486 or forced into standard mode with WIN/S on a 386; or you can run it on a 286. The Windows 3 DPMI host does test and return the proper CPU type (presumably, it uses self-modifying code and the differing lengths of the prefetch queue). The DPMI entry point is always at selector F000h, but the offset varies by machine and invocation.

THE OS/2 API

As mentioned above, Phar Lap took the easy path by supporting two APIs. If they had supported only the DOS interrupt

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug C

What's *your* problem?

- *program hanging?*
- *memory overwrites?*
- *operating system overwrites?*
- *corrupted heap?*
- *memory leaks?*
- *dangling pointers?*
- *open file pointers?*
- *invalid calls to free()?*

If your code has a bug that's making *you* sick, *call the doctor*—**Dr. MD, the run time memory debugger for C!** Sure, you *could* find it yourself, but how much would it cost you in time, money and aggravation?

Dr. MD is the **easy and cost effective** way to cure your memory corruption ills. **Dr. MD** gives a **complete diagnosis** including the source file and line number of your bug.

- *includes source code*
- *non obtrusive*
- *easy to customize*
- *hardware independent (386 not required)*
- *operating system independent (DOS, UNIX, OS/2...)*
- *small library*

Start your code on the road to recovery—order your copy today!

DR. MD™

NEW version 2.0
$129.00 plus shipping

RSS Engineering
Custom and Contract
Software Solutions

MasterCard VISA

2683 N. Taft
Loveland, CO 80538 USA
(303) 669-3300
FAX (303) 669-3354

RSS Engineering
a division of
Remote Switch Systems, Inc.

21h API, Phar Lap developers would have been busy testing, disassembling, and patching the DOS (real-mode) C runtime libraries so that they would execute properly in the protected mode. And every minor compiler and library update would have required this kind of effort. This is precisely what the other 286 DOS-extender vendors (Rational and Ergo) have been doing for the past four years. Of course, these other products support many more compilers and development environments.

By adding the OS/2 API, Phar Lap was guaranteed code generation, as well as runtime libraries that should run, mostly without change, in the protected mode. Even here they hit a few snags, and there are still some library patches. For example, QuickHelp, PWEB (the workbench), the Resource compiler, and a few other OS/2 executables do not run under RUN286. The new Microsoft Profiler (OS/2 version) also makes a few **DOSPTrace** calls not currently supported by RUN286, but they will be in an update. All in all, the 286|DOS-Extender is definitely an interesting product.

ERGO EDU-16 FOR DPMI

Ergo Computing is about to ship its latest DOS extender, EDU-16 (Ergo DPMI Utilities for 16-bit applications). EDU-16, a rewrite from the ground up, takes full advantage of the DPMI interface under Windows; for non-Windows environments, Ergo still offers its OS/286 and OS/386 DOS extenders. EDU-16 is the only DOS extender that can function as a DPMI host *or* a client, so you can run my little DPMI inquiry program under EDU-16 stand-alone, and it'll return information on Ergo's implementation of DPMI. When running an application under Windows in the enhanced mode, EDU-16 uses the Windows DOS extender; under DOS, it uses the Ergo kernel.

The EDU-16 development kit ($495) includes the DOS-extender kernel, replacement **obj** files for various popular compilers, a symbolic debugger, a program loader, and example code. The kit supports development with Borland C++, Microsoft C 5.1 and 6, and Fortran 5. Borland's BGI is also supported.

Although EDU-16 runs segmented executables (as do Windows and OS/2), it supports the familiar DOS (INT 21h) and ROM BIOS interfaces. EDU-16 includes a library of DPMI functions callable from C or C++ and supports the creation of DLLs that can be marked for either real- or protected-mode operation. The kit is particularly attractive for Borland C++ users, because the Borland IDE, including the IDE linker, can be used for development.

Running under Windows in standard mode, EDU-16 has the same limitations as the other products. Virtual memory is available in enhanced mode. If your application is stand-alone and won't be running under Windows, Ergo's other DOS extenders (OS/286 and OS/386) are available with virtual memory options.

When you're ready to ship your product, you buy an unlimited redistribution license for a reasonable, one-time fee ($500), and Ergo supplies utilities to bind the EDU-16 kernel with your application for distribution. Ergo is working on a 386 version of this product (EDU-32) for future release.

RATIONAL'S DOS/16M

Rational Systems' DOS/16M 286 DOS extender has been around for years. Being bundled with Lotus probably gives it more users than any other DOS extender. The current version (v3.95) supports DPMI, so it can also run under Windows 3. DPMI 0.9 and 1.0 are both supported, but without a DPMI 1.0 host, Rational is reluctant to call it 1.0-compatible. DOS/16M is a close parallel of the INT 21h API of DOS. Its support of many extended functions makes it easier to move to the protected mode.

Late this spring, Rational released Oxygen, a product for Microsoft C 6 that binds a 16-bit OS/2 API DOS extender to the C 6 OS/2 executables so they can run under DOS. This keeps memory-hungry C 6 from running out of heap space under DOS. They had this OS/2 technology in house for about 18 months. Unfortunately for Rational, this release coincided with Microsoft's bundling of its Windows DOS extender with C 6.0ax.

When running under its own DOS extender, DOS/16M supports virtual memory by swapping out segments to disk. Under Windows 3, it acts as a DPMI client and runs in a DOS box. Under Windows 3 enhanced mode,

Circle 22 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

large memory spaces and virtual memory are available. If it is run under Windows standard mode, the Windows 3 limitation discussed above applies.

DOS/16M SDK ($5,000) supports a number of compilers as development environments: Microsoft's C, Fortran, and Pascal; IBM's C/2 and Pascal; Borland's Turbo C; Watcom C, Lattice C, Logitech Modula 2, JPI Modula 2, and Zortech C++. For these compilers, replacement object modules are provided for their runtime libraries. The supported compiler is often not the latest version. DOS/16M is strong on national language support and protected-mode TSRs. Rational supplies a Windows virtual device driver to allow TSRs developed with DOS/16M to work properly under Windows. The SDK also includes a protected-mode linker and symbolic debugger for DOS-extender apps.

During testing, you can use Rational's loader, but when your application is complete, the development kit includes various source and **obj** modules to splice the loader to your application. DOS/16M is intended to be an OEM product, and its price reflects this fact.

HAPPENINGS AT THE HIGH END

The real action in the DOS-extender world is at the high end, and the action is shifting to the compiler vendors. For much of this material I am under nondisclosure agreements and can only comment on generally available information.

Although Intel has been marketing its Intel 386/486 C Code Builder Kit ($695), it has been withholding the product from reviewers. You can still buy it from several mail-order suppliers, though. The Intel kit includes a 386 C compiler (developed in-house), a linker, a symbolic debugger, a make utility (the excellent Opus Make), and a new version of the IGC DOS extender.

The DOS extender supports DPMI, so it runs under Windows but does *not* currently support VCPI. You can't run the DOS extender under QEMM, 386-MAX, or other memory managers, which is a drawback. This is due to the design of the IGC DOS extender: it's the only product that runs DOS in the virtual-8086 mode instead of switching to real mode.

The Intel kit is easy to use, and the C compiler library is Microsoft-compatible. When you run the compiler driver (icc), it automatically links your application and produces a protected-mode executable. It comes with a 387 emulation library, so a math coprocessor is not required. The enticement for small developers is that there is no charge for bundling the DOS extender runtime with your product; one box price gets everything.

But the current version is not without flaws. First, it's the only DOS extender that is incompatible with VCPI. When running under Windows 3 DPMI, the DOS extender has some nasty bugs related to Control-C behavior and virtual memory. The C compiler fails some ANSI validation tests, including the Plum Hall sampler. Intel is working to fix these problems.

The exciting news is that Silicon Valley Software (SVS) will also bundle the Intel (IGC) DOS extender with its fine line of C, Fortran, and Pascal compilers—again, with no runtime fee and priced reasonably. The Pascal and C compilers are priced at $325, and the Fortran compiler is $395. My experience with SVS compilers has been very favorable. They generate very fast code, particularly for numerics. This combination is something to watch for.

Phar Lap's v4.0 of 386|DOS Extender now supports DPMI. You can use this DOS extender for creating applications that will run stand-alone or under Windows 3 enhanced mode. This does *not* mean that they will be Windows apps, but they can execute in a Windows 3 DOS box.

Rational Systems is poised to release its first 386 DOS extender, DOS/4G. Like DOS/16M, DOS/4G is DPMI-compatible, and it comes with Rational's protected-mode linker and debugger. It supports the DOS INT 21h API and works with MetaWare's High C/386 and Watcom's C/386, as well as with Microsoft's OS/2 2.0 C32 compiler.

The Heap Expander® Gets Big with V.3.0

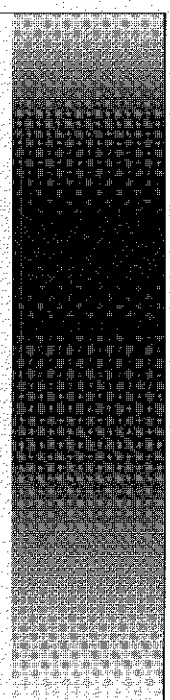
A Major Upgrade for the Groundbreaking Memory Management Library

•**New Link Library** accesses expanded memory, extended memory, conventional memory and hard disk in any combination.

•**Supports** XMS and VCPI standards for extended memory and LIM standard 4.0 for expanded memory. Supports DOS 4.0, 5.0 and most standard C and C++ compilers.

•**Backward Compatible** with version 2.0.

Also, **PowerSTOR®** v. 1.3 now available supporting VCPI standard for extended memory. Completely compatible with expanded memory emulators.



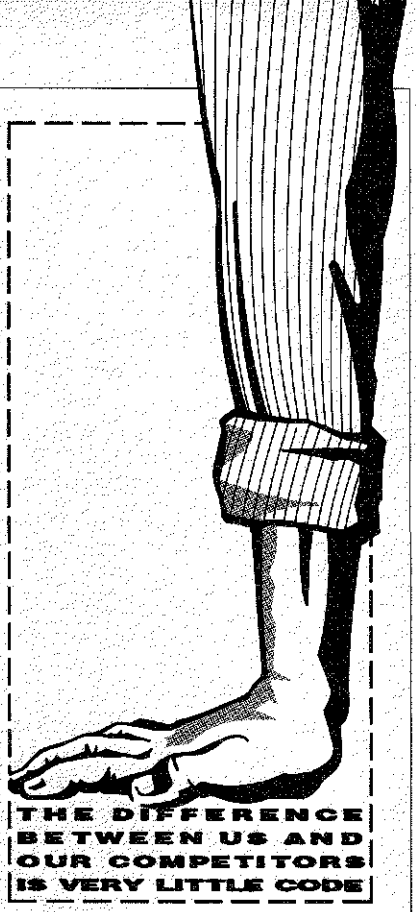
Call Toll Free to order
USA VISA / MC
800-248-1045 ext 100
For more information, call The Tool Makers direct or Fax

Either Program Just **$119.95** each

THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690
408-458-1011

Circle 23 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Magic TSR Toolkit

Write complete memory resident programs that require as little as 2K of RAM by swapping to EMS or disk. No secrets: complete assembly language source code and tutorial provided

Quantasm Power Lib

- Over 400 fast, compact assembly language routines
- Overlapping windows in less than 3K
- Over 100 string and numeric functions
- Menus, date, time, sound and more!

Magic TSR Toolkit . . . $199.95
Power Lib. 99.95
Power Lib w/source . . 299.95
ASMFLOW. 99.95

Shipping & Handling: U.S. $5; Foreign $15
CA res. add sales tax
VISA and MasterCard accepted
30-day money back guarantee

QUANTASM CORPORATION

19855 Stevens Creek Blvd., Suite 154
Cupertino, CA 95014
408/244-6826

VENDOR INFORMATION

EDU-16, OS/286, and OS/386
Ergo Computing
One Intercontinental Way
Peabody, MA 01960
(508) 535-7510
Circle 219 on Reader Service Card

Intel C Code Builder Kit and DPMI
Intel Corporation
P.O. Box 58130
Santa Clara, CA 95052-8130
(800) 548-4725
Circle 220 on Reader Service Card

Windows Applications Developer Kit
MetaWare
2161 Delaware Avenue
Santa Cruz, CA 95060
(408) 429-6382
Circle 221 on Reader Service Card

NDPWIN C, C++, Fortan, and Pascal
MicroWay, Inc.
Box 79
Kingston, MA 02364
(508) 746-7341
Circle 222 on Reader Service Card

OTG Fortran and DBOS
Distributor: OTG Systems, Inc.
P.O. Box 239
Clifford, PA 18413-0239
(717) 222-9100
Circle 223 on Reader Service Card

286|DOS-Extender; 386|DOS-Extender
Phar Lap Software
60 Aberdeen Avenue
Cambridge, MA 02138
(617) 661-1510
Circle 224 on Reader Service Card

SVS 386/FORTRAN 77, 386/ANSI-C, and 386/Pascal
Silicon Valley Software (SVS)
1710 South Amphlett Blvd., Suite 100
San Mateo, CA 94402
(415) 572-8800
Circle 225 on Reader Service Card

Watcom C/386 and Fortran 77/386 for Windows
Watcom
415 Phillips Street
Waterloo, Ontario
Canada N2L 3X2
(800) 265-4555, (519) 886-3700
Circle 226 on Reader Service Card

DOS/16M and DOS/4G
Rational Systems
220 North Main Street
Natick, MA 01760
(508) 653-6006
Circle 227 on Reader Service Card

Salford Software's new WDBOS v2.60 DOS extender now supports DPMI. This British import is bundled with OTG Fortran compilers and with several other development tools from Great Britain. A soon-to-be-released 386 Pascal from Sheffield also uses WDBOS. WDBOS is a TSR with a small DOS footprint; its library of routines resides in protected mode. The runtime libraries for the OTG 386/486 Fortran compilers actually reside in the WDBOS DOS extender. Pricing for the earlier DBOS unlimited runtimes is $495.

MicroWay has released a very interesting product for 386 developers. NDPWIN, designed for versions of its 386/486 compilers (C, C++, Pascal, and Fortran), runs under Windows 3 and allows you to develop applications that run under Windows 3. NDPWIN is a custom loader that runs as a Windows application. This in turn sets up a 386 flat model. The NDPWIN language libraries interface directly both with the WINMEM32 DLL supplied with the Windows SDK and with the low-level Windows 3 I/O functions (**OpenFile, _lread,** etc). Again, these are not true Windows apps; they're 386 character-based apps that run as Windows apps under Windows 3 enhanced mode. No separate DOS extender is necessary for redistribution. (By the way, *PJ* contributor Mark Barrenechea has been named vice-president for language development at MicroWay.)

TRUE GUI APPS

So far I've talked about running 286 or 386 DOS-extender applications under Windows 3. This is a convenience to users who prefer Windows as their multitasker or menu interface to DOS. But the real challenge is developing true 32-bit Windows GUI applications that can support the full Windows API. Both Watcom and MetaWare are about to release products that accomplish that feat. Rational has also announced a similar product called BigWin, which is designed to link with several 386 C compilers.

The Microsoft Windows API is a large, 16-bit interface, and a number of functions issue call-backs to your code. Watcom and MetaWare have had to build a glue layer (they both call

See Baker, page 90

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Managing memory

The "whys and wherefores" of the Macintosh Memory Manager

By Brett Glass

Most early microcomputers simply did not have memory management. CP/M-80 systems, for example, ran almost exclusively in a simple, flat address space consisting of 64K or less of RAM—all visible to the processor at once. (Some CP/M computers did provide bank-switching capability. It was used most often for "phantom" boot ROM or to hold the operating system so that the Transient Program Area could be as large as possible.)

By contrast, nearly all the latest and most powerful microprocessors (e.g., the 80486 and 68030) have memory management units (MMUs) built in, so they can handle mapping, paging, and virtual memory with little or no additional hardware.

The Macintosh Memory Manager, introduced with the original Macintosh computer in 1984, represents an intermediate step between these worlds: a set of operating system routines and programming conventions that provide flexible, relatively efficient memory allocation in a system that lacks hardware memory mapping or an MMU. This article delves into the "whys and wherefores" of the Macintosh Memory Manager.

WHY A MEMORY MANAGER?

If the CP/M-80, Apple II, IBM PC, and other popular systems of the day didn't require sophisticated memory management software, why did the Macintosh? One reason was economy. To maintain its fabled 50% gross margin on hardware and still sell a computer that users could afford, Apple could put only 128K of RAM into the Macintosh—in retrospect, too little to handle the requirements of the operating system, GUI, and application software. The basic system resources—including fonts, code, bitmaps, and other large data structures—could easily overflow memory if they were all present at once. What's more, code for the 68000—the CPU at the heart of the Macintosh—consumes more space per operation than equivalent code on many other CISC processors. This made the Mac's memory crunch far more acute than on some of its less ambitious character-based contemporaries, which at the time typically sold with 256K of RAM.

A hardware MMU similar to that on the early Sun workstations (a clever design that came out around the same time) would have been too expensive.

Brett Glass is a freelance programmer, author, hardware designer, and perpetual student residing in Palo Alto, California. Brett was one of the original architects of the IEEE 802.5 Token Ring LAN, and he co-authored Living Videotext's ThinkTank 2.0. Brett earned his MSEE from Stanford University and his BSEE from the Case Institute of Technology. He can be reached as [72267,3673] on CompuServe or as "rogue@well.sf.ca.us" on USENET and the Internet.

FIGURE 1: Simple 10K heap

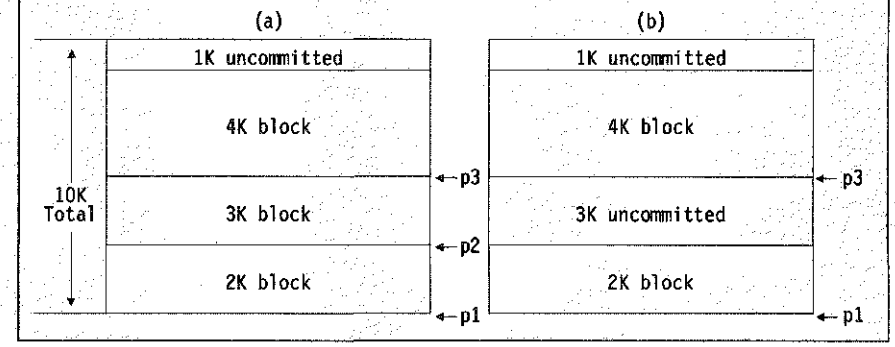


ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Macintosh opted to perform memory management in software instead and chose a flat physical address space—the simplest and cheapest to implement. These factors all made an efficient memory manager vital to the success of the Macintosh.

FRAGMENTATION WOES

Before we examine the specifics of the Macintosh Memory Manager, let's look at the techniques used for managing a simple *heap*—a pool of dynamically allocated memory. Suppose we start with a contiguous block of uncommitted RAM (a *heap zone*, in Apple parlance) 10K in size. An application begins to request memory, asking for blocks of 2K, 3K, and 4K (Figure 1).

In each case, a call such as **new** or **malloc** (depending on the language) returns a pointer to a block of memory the application can use. Most heap implementations use a *first fit* algorithm to find the memory to be allocated; they start at one end of the heap zone and allocate the requested space from the first chunk of uncommitted space that's larger than or equal to the required size.

Figure 1b shows what happens if the 3K block is freed, leaving 4K of available space. There's now 4K available, but it's discontiguous; a request for a 4K block of RAM would fail. This phenomenon, called *fragmentation*, can be costly. In a system that's been running for a long time, large amounts of memory can be wasted, because small fragments of memory are caught between other blocks.

The example is an oversimplification, of course; we haven't shown the headers and trailers that most heap managers attach to blocks. But the problem of fragmentation is endemic to heaps in general.

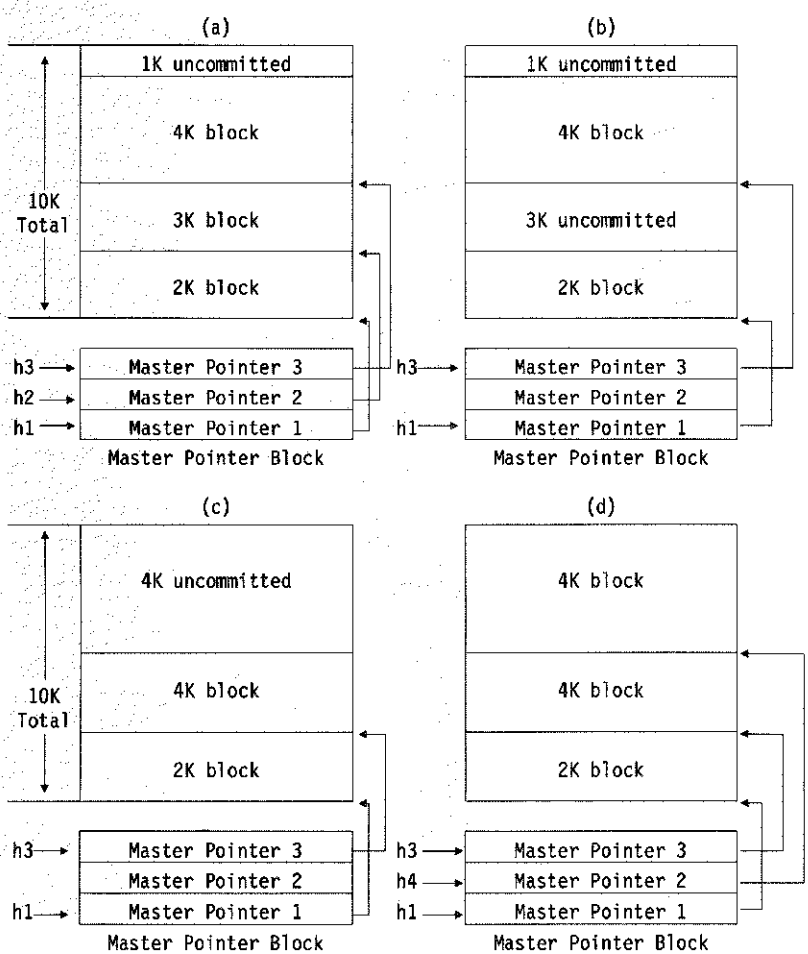
An MMU enables a system to handle fragmentation easily. Logical addresses can be mapped to physical addresses, so discontiguous blocks of RAM can "look" contiguous to an application (subject to granularity constraints). But the Macintosh, with its unaided 68000 CPU, couldn't take this approach. What about consolidating the contents of the discontiguous blocks by moving them around physically? Unfortunately, although the 68000 can do this, there's not any way to update all the application's pointers to point to the correct places after the move. And failure to do so would, as you might expect, result in dangling pointers and almost certain disaster.

Because neither of these approaches was feasible, the Macintosh took a third: a technique called *double indirection*. An application that requests a block of memory is given a data object called a *handle* instead of a pointer to the memory. A handle is a pointer that points to another pointer, called a *master pointer*, which in turn points to the memory.

How does this tactic help to solve fragmentation problems? Figure 2 shows the scenario described above, but adds double indirection. Note that it's possible to change a master pointer without leaving any of the application's handles "dangling." This means that the memory manager can compact the heap by moving blocks; it only has to keep the master pointers current.

This, in essence, is the way the Macintosh Memory Manager allocates relocatable blocks. (We'll come back to a few additional housekeeping details.) When the application calls the memory manager's **NewHandle** routine, it reserves memory, sets up a master pointer, and returns a handle (the address of the master pointer).

FIGURE 2: Heap with double indirection



USING THE MEMORY MANAGER

With the power of double indirection comes responsibility: the programmer must know and understand the conventions that allow handles to work properly.

Suppose that, while programming the Macintosh, you wanted to allocate space on the heap for an integer variable. (In practice, you'd probably

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

want to allocate something larger, but let's keep things simple.) You might reserve the space in the following way:

```
type
  IntPtr = ^Integer;    {A pointer to an integer  }
  IntHandle = ^IntPtr;  {A handle (a pointer to   }
                        {a pointer) to an integer }
var
  i : IntHandle; {Variable to hold the handle     }
                 {for the allocated memory        }
...
  i := IntHandle(NewHandle(SizeOf(Integer)));
```

NewHandle takes a long integer argument and returns a value of the type **Handle.** (**Handle** is defined in the Macintosh headers as a pointer to a pointer to a signed byte; you must use a typecast to assign the value to a handle that references something else.) If sufficient memory is not available, the handle is NIL; otherwise, it can be dereferenced twice to get at the allocated memory.

Note the use of **SizeOf** and a typecast in the example. **SizeOf,** which returns the size of an object of a given type in bytes, is a good way of making sure you allocate the right amount of space. It's necessary to cast the result of **NewHandle** to the type **IntHandle** to tell the compiler that it's OK to perform the assignment.

To access the variable you created, simply dereference the handle twice, like this:

```
i^^ := 2;     {Assign a value to a heap variable}
Writeln(i^^) {Then print its value}
```

In an ideal world, each application would perform a double dereferencing operation on a handle every time it accessed data on the heap. This would take changes in the master pointer into account at all times. In real applications, however, speed definitely matters. The extra memory reference needed to fetch the master pointer would slow things down enormously—a burden that high-performance software can ill afford.

At first glance, it appears that you might be able to speed things up by dereferencing the handle once at the beginning of the routine and again each time you used the variable, as follows:

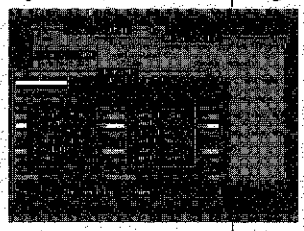
```
var
  masterPointerCopy: IntPtr; {Copy of a master  }
                             {pointer to an integer }
...
masterPointerCopy := I^;  {Dereference the      }
          {handle once at the start to get a    }
          {copy of the master pointer           }
...
masterPointerCopy^ := 2; {Defererence again when}
                         {using the variable    }
```

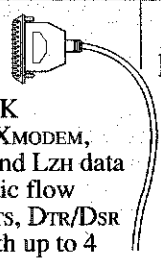
But this technique can be dangerous. Suppose that, between the first and second assignments above, the operating system compacted the heap and updated the actual master pointer. The variable **masterPointerCopy** would not be updated and would no longer point to the correct place in memory; the program would therefore generate erroneous results.

For this reason, the Macintosh provides routines that "lock" a block of memory—that is, they prevent the memory manager from moving it and

OOP Async Tools for Turbo Pascal 6.0 with ZMODEM and Data Decompression

Async Professional is a full-featured object-oriented communications toolkit that enables you to build powerful async applications faster and easier. Async Professional has all the standard features you'd expect of an async toolbox. Plus the robust ZMODEM transfer protocol, data decompression tools, and both object-oriented and procedural calling interfaces.





You get ■ interrupt driven buffered I/O to 115K baud ■ ZMODEM, Kermit, XMODEM, YMODEM protocols ■ ZIP and LZH data decompression ■ automatic flow control–XON/XOFF, CTS/RTS, DTR/DSR ■ COM1- COM8 support with up to 4 ports open at once ■ 16550 buffered UART support ■ hardware interrupt sharing on PS/2 ■ ANSI terminal emulation ■ modem control including Hayes, V.32, V.42, and MNP5.

Hot Example Programs Too!

Demo programs include a powerful multi-window comm program with menus, text editor, and mouse support, built using Object Professional, and a simple comm program that depends on no other products.

Powerful Async Debugging

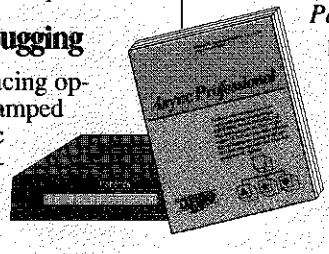
Event logging and tracing options create a time-stamped audit trail of all async interrupts and characters sent or received. You'll debug your application faster without the need for special debugging hardware.

Full Source, Documentation

Async Professional includes complete documentation, pop-up help, free technical support on CompuServe and by telephone direct from the authors. You get full source code and pay no royalties.

"Async Professional is an excellent series of routines...and very complete. It's better than anything I've seen in Pascal, C or assembler."

Richard Wilkes, Author, TAPCIS



Async Professional, only $139.

Call toll-free to order. 1-800-333-4160

Satisfaction guaranteed or your money back within 30 days. Add $5 per order for standard shipping in U.S./Canada. Inquire about other shipping options. Includes both 5.25" and 3.5" disks. Registered LiteComm owners call for special upgrade offer. OOP units require Turbo 5.5 or 6.0; non-OOP units require Turbo 5.0 or later.

TURBO Power

9AM–5PM MST Monday through Friday, USA & Canada.
For more information call (719) 260-6641, fax to (719) 260-7151, or send mail to CompuServe ID 76004,2611
TurboPower Software PO Box 49009 Colorado Springs, CO 80949-9009
© TurboPower Software 1991

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

changing its master pointer. The call **HLock(i)**, made before the master pointer was copied, would ensure that the block would not be moved. The call **HUnlock(i)** would let the block be moved once again.

Locking a block can keep the memory manager from fully compacting the heap. For instance, if the 4K block in Figure 2b had been locked, the compaction process could not have freed up enough memory to honor the next request for memory.

If an application needs a chunk of memory for a quick calculation, it can allocate a nonrelocatable block, do its work, and then deallocate the block right away.

When dealing with handles, you must be careful of language constructs that implicitly dereference them. For instance, an efficient compiler, given the Pascal statement

```
with theHandle^^ do
  begin
  ...
  end;
```

(where **theHandle** is used to access a record variable), will generate code that dereferences the handle, saves a copy of the result in an address register, and then uses that copy plus an offset to access each field of the record. To allow the compiler to do this with impunity, make sure that the handle is locked while the **with** statement is executed.

PURGEABLE BLOCKS

As mentioned above, the Macintosh operating system and applications frequently use *resources*—chunks of code and data (such as fonts) that are loaded from disk when needed. If all this data needed to be in RAM at once, little (if any) memory would be left for other things. Fortunately, because copies of these data structures reside on the disk, no information is lost if they're removed, or purged, from memory—freeing the space they occupied—to honor an allocation request. A heap block that contains such expendable data is called *purgeable*.

The memory manager purges a block from memory only as a last resort—that is, when compacting or expanding the heap space won't work. When the memory manager purges a block from memory, it sets the master pointer for that block to NIL. When the application program or the operating system can see that this has occurred (that is, that the handle is *empty*, in Macintosh parlance), it can ask the memory manager to reallocate some space for that handle using the **ReallocHandle** procedure; the program can then reload the resource.

For safety and convenience, all blocks on the heap start out unpurgeable. **HPurge** makes a handle purgeable, and **HNoPurge** reverses the process. A program can force a block to be purged by calling **EmptyHandle** with the handle as a parameter.

NONRELOCATABLE BLOCKS

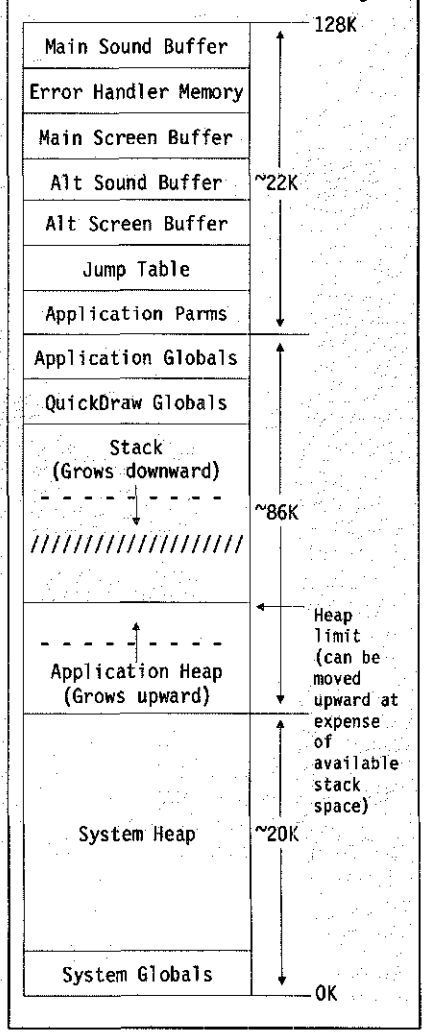
If an application wants to allocate some memory for especially fast access, it can ask the memory manager to allocate nonrelocatable blocks, as in a simple heap. These blocks are allocated via pointers that point directly to them, rather than via handles; they do not have master pointers associated with them. The function that allocates a nonrelocatable block is **NewPtr**.

Like locked blocks, nonrelocatable blocks can't be moved and may cause fragmentation. There are several ways to minimize this problem. If an application reserves its nonrelocatable blocks at the start of execution, before asking for any relocatable ones, the nonrelocatable blocks will be allocated contiguously at the bottom of the heap. No fragmentation occurs unless one of these blocks is subsequently freed. This approach is ideal when the memory will be used throughout the program's lifetime.

Master pointers always reside in nonrelocatable heap blocks. In the Mac's *application heap* (the heap zone your application usually uses), each such block—called a *master pointer block*—contains 64 pointers. This is exactly the kind of block to reserve early if you think you might need it. Why? Although the memory manager can create master pointers automatically at any time, it may cause fragmentation when it creates new, nonrelocatable master pointer blocks. Therefore, all but the most trivial applications should call the procedure **MoreMasters** one or more times upon start-up. Each call creates an additional 64 master pointers for use by the memory manager.

Nonrelocatable blocks need not be allocated only at launch time. If an application needs a chunk of memory for a quick calculation, it can allocate a nonrelocatable block, do its work, and

FIGURE 3: "Classic" Macintosh RAM layout



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

then deallocate the block right away. This technique is equivalent to creating a relocatable block and locking it, but it's much more efficient: the system doesn't have to set up a master pointer, and the program doesn't have to explicitly lock and dereference a handle.

THE BIG PICTURE

Figure 3 shows how RAM was used in the original 128K Macintosh (other members of the Macintosh family are similar). Note that there are two heap zones: the application heap and the system heap. Each is usually used for a specific set of functions, and most applications use only the application heap; however, they can use the system heap if necessary.

When a program is launched on the Macintosh, a chunk of memory is allocated to hold its stack and its application heap. The stack grows toward lower addresses, and the application heap—initially given an upper limit by the operating system—can be grown toward higher addresses by raising the limit. A low-level operating system routine checks the two periodically to make sure they stay a safe distance apart.

Under MultiFinder (see below), things are a little more complex. Each application has its own stack and its own application heap, and they don't start at opposite ends of free memory. But the essential concepts are the same in all cases.

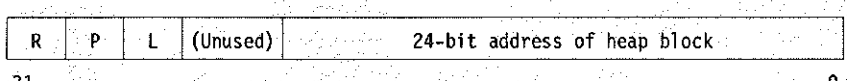
KEYS TO COMPATIBILITY

As the Macintosh architecture grows and changes, fewer and fewer assumptions can be made about the internals of the system. Assumptions can make programs fail unexpectedly, and sometimes catastrophically.

One source of problems for some of the Macintosh applications was the advent of MultiFinder, Apple's cooperative multitasking environment. In this environment, a program could no longer assume that it "owned" the machine—now it needed to share it with other applications.

Some of the new rules invalidated the tactics that had been used by certain applications to manage memory. For instance, a program couldn't plan its memory use by "scoping out" the machine's resources to find out how much memory was there. (If it did so, another application might not be able to run or might take some of the available memory without warning.) Instead, an application could specify only the minimum it needed to run—and the maximum it would ever want. Likewise, it became unwise for an application to use the system heap for its own storage, especially nonrelocatable storage. Doing so could clutter up memory that was needed to serve all the programs on the machine.

FIGURE 4: A master pointer in the original Macintosh



R = Resource bit (1 if block is a resource)
P = Purge bit (1 if block is purgeable)
L = Lock bit (1 if block is locked)

32-BIT CLEANLINESS

Another key to compatibility is making code "32-bit clean"—that is, using the upper byte of an address only for addressing. The reason is historical: the 68000 microprocessor, released in 1981, faced limits on package size, chip size, and power consumption that prevented it from allowing users access to 32 address lines. Only the 24 least significant bits of an address left the chip, so the upper byte of a pointer could be used for other things.

Later processors in the 68000 family remedied this shortcoming; the 68020 had a full 32-bit address bus. But the Macintosh ROMs had already been coded to place flag bits in the highest byte of many internal pointers, and the memory manager was among the biggest culprits.

Figure 4 shows the layout of a master pointer as implemented in the original Macintosh. Note that the uppermost byte contains three flags: the *resource bit*, the *purge bit*, and the *lock bit*. The resource bit is set by the Resource Manager to mark a block containing a resource. The lock bit is set if the block is locked, and the purge bit is set if the block is purgeable.

The Macintosh designers had anticipated the possibility of moving to processors with 32-bit addressing, and they provided routines (such as **HLock, HUnlock, HPurge,** and **HNoPurge**) to manipulate these flags. This, they reasoned, would allow them to move the flag bits to some other location in the future. But early copies of *Inside Macintosh* documented the exact locations of the flags and even provided assembly language symbols for them, encouraging programmers to save a system call and manipulate the flags themselves.

Because the original memory manager—along with many application programs—used the upper bytes of pointers in this and similar ways, the Macintosh II hardware was set up to mask the high byte of every address unless a specific bit in the Versatile Interface Adapter (VIA) chip was turned on. (Ironically, this is the same approach IBM took in the IBM PC/AT, where the A20 address line could be disabled to make the 80286 look more like an 8088.) But the Mac II's physical (and logical) address space was still limited to 16MB, except in device drivers and some other system code.

APPLE COMES CLEAN

With System 7.0, Apple has at last renounced its "unclean" ways. System 7 moves the flags out of each master pointer, and a systemwide option allows programs to use a full 32-bit address space. Unfortunately, on some of the older Macs—the II, IIx, IIcx, and SE 30—the ROMs themselves still aren't 32-bit clean. On these machines, using System 7 doesn't give you the option of 32-bit mode.

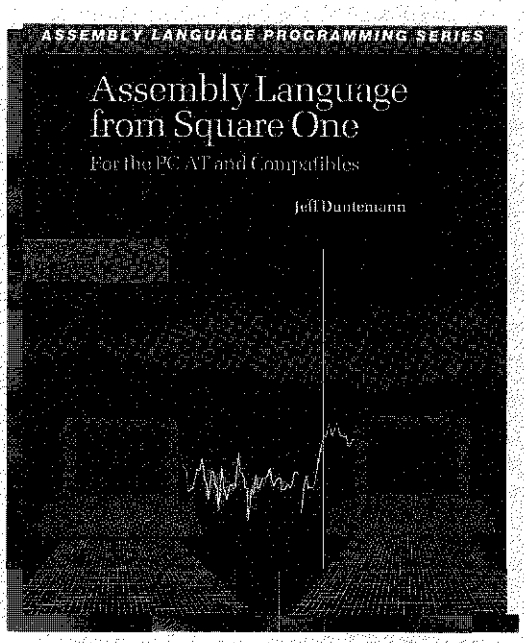
Fortunately, a company called Connectix has a workaround. For $169 or less, you can buy an operating system extension called Mode32. The extension patches around parts of the older ROMs and lets those systems enter

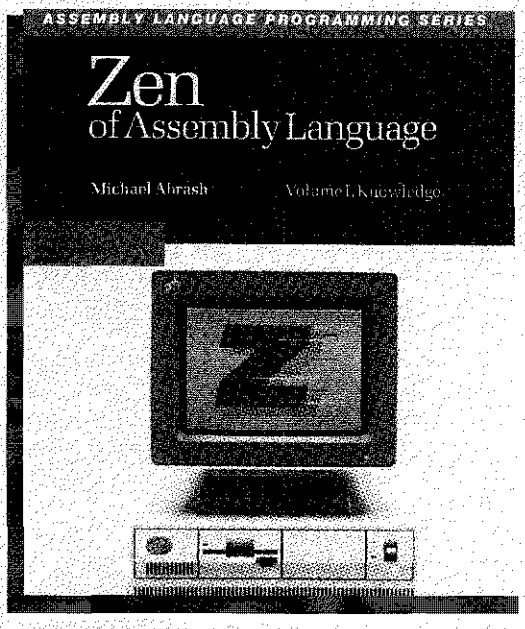
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Best Books from PC TECH

Take It To The Limit!

Jeff Duntemann taught you object-oriented programming in the pages of *PC TECHNIQUES*, and he can teach you assembly as your first programming language. Jeff's crisp metaphors and offhand, humorous style carries even the very toughest concepts home. After reading this book you'll be writing programs in assembly, but more important, you'll know both what you're doing and why you're doing it. **Every copy signed by the author!**





Order *Assembly Language from Square One* from PC TECHNIQUES and receive the listings diskette for FREE!

"Pushing the Envelope" columnist Michael Abrash has made hand-optimized assembly code his trademark, and in this monumental volume shares the accumulated wisdom of his entire career as a performance programmer. Understand how the 80x86 family does what it does. Learn of the "cycle eaters" that slow your code down. Learn how to do what no higher-level language compiler can do. That's the Zen of assembly language. **Every serious programmer must have this book!**

Send Coupon or Call Today!—602-483-0193

Assembly Language from Square One $29.95 (Free Disk)

Zen of Assembly Language $29.95

Subtotal $_____

Shipping:
$2.50 per book in US.
$5.00 per book outside US.

$_____

Total $_____

Prepayment required—U.S. Banks only

___Check/Money Order ___VISA/Mastercard

Card #_____ Exp. Date_____

Name_____

Company_____

Address_____

City/State/Zip_____

Country_____

Phone_____

PC TECHNIQUES, 7721 E Gray Rd, Suite 204, Scottsdale, AZ 85260

STOP!

Don't write another line of assembly code until you've read these books!

Circle 26 on Reader Service Card

PRODUCED 2004 BY UN2.ORG
ELECTRONIC REPRODUCTION PROHIBITED

32-bit mode. The Mac's control panel lets you go back to 24-bit mode for the few applications that don't work with 32-bit mode.

LET'S GET VIRTUAL

System 7 also provides *demand-paged virtual memory* on processors that support it, using either an external MMU (the 68851) or internal MMUs (the 68030 and 68040). Before System 7 shipped this spring, the only way to get virtual memory was to buy a program called Virtual, from Connectix. While System 7 was still vaporware, this company forged ahead with a utility that let users enjoy the benefits of virtual memory (it's still useful if you're running an earlier version of the Mac operating system). Also, unlike System 7, Virtual provides virtual memory on the 68030 accelerator boards that fit inside the Mac Plus, SE, and Classic.

Virtual is an *init*—a utility that loads beneath the Macintosh system software at boot time. The OS and applications can't tell that anything has changed, but they can now "see" up to 13.75MB of RAM in the machine. Virtual uses the MMU's paging unit to swap data to and from disk as needed, keeping only the most recently used data in physical RAM. (This is often called a *least recently used* [LRU] algorithm, because it picks data that hasn't been accessed for a while to move to disk.) Code and data are swapped in 4K chunks called *pages*; if the program attempts to access something that's not in physical RAM, a *page fault* occurs and special code brings it in from the disk.

VIRTUAL IS ITS OWN REWARD

An obvious advantage of virtual memory is that programs that once ran out of memory will now be able to execute—albeit more slowly. Even if an application worked adequately before, it may work more smoothly with virtual memory.

How can this be? One reason is that the LRU algorithm frees the programmer from a tricky task: deciding which parts of a program should be in memory at a given time. On the original Macintosh, programmers had to break their programs up into segments and use a part of the Macintosh OS called the *segment loader* to control which ones were in RAM at any given time. Unfortunately, there were so many factors to consider (such as sizes of fonts and system resources, desk accessories, the state of the Clipboard, the amount of available RAM, and possible commands the user might invoke) that programmers often chose unwisely and inadvertently slowed things down instead of speeding them up.

The LRU algorithm frees the programmer from a tricky task: deciding which parts of a program should be in memory at a given time.

An LRU algorithm avoids this problem by using the program's recent history to determine which parts of memory are least likely to be needed in the future. The programmer can still use the Segment Loader to provide "prescience"—that is, to give the system a hint about what will really happen next—but no longer has to oversee things as carefully.

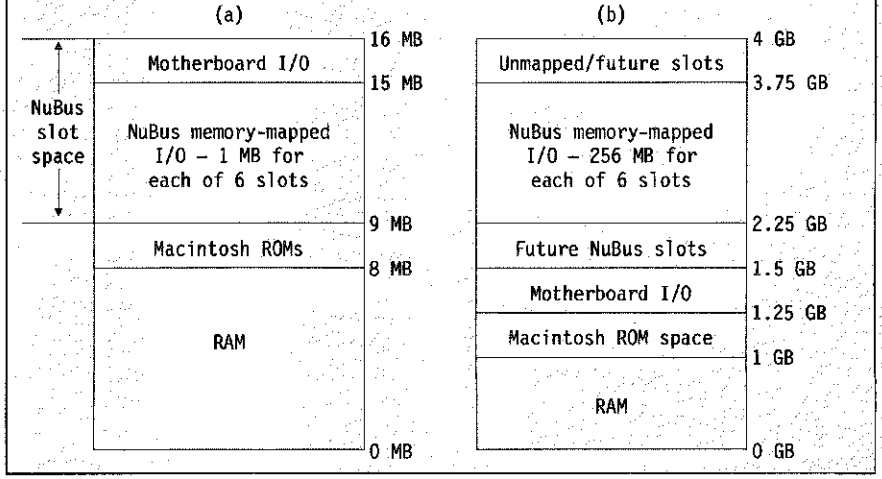
Virtual had to overcome another knotty technical problem. It couldn't swap out its own code (of course), nor could it swap out the disk routines that do the swapping, the interrupt service routines, or the data used by any of this code. Clearly, an intimate knowledge of the Macintosh ROMs and OS was required.

BEYOND THE 8MB BARRIER

Even though 24-bit addresses give the Macintosh an address reach of 16MB, not all of this space was originally intended to be RAM. The early Mac designers, not anticipating users' future memory needs, expected that only the lower 8MB of the address space would ever be used for RAM. Figure 5a shows the layout of the 24-bit address space. Note that each slot on the Mac II's NuBus was allocated a full megabyte for use in memory-mapped peripherals. (The motherboard, which was technically considered to take up one slot, mapped its peripheral chips into the very highest addresses.)

The first version of Virtual made no changes to the operating system traps and calls; it could fill in the area below 8MB only with virtual memory. Using the next megabyte of address space would hide the Macintosh ROMs (whose addresses are hard-coded to start at 8MB) from the rest of the system. This was useful if you had not fully populated the motherboard on your Mac II, but if you already had 8MB of physical RAM, no contiguous address space was left for virtual

FIGURE 5: 24-bit and 32-bit Macintosh address spaces



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk and Manual only $75 US *Call for prices for*
AMX 86 $3000 US *other processors.*
(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 27 on Reader Service Card

VENDOR INFORMATION

Mode32 and Virtual
Connectix
125 Constitution Drive
Menlo Park, CA 94025
(800) 950-5880
Circle 213 on Reader Service Card

memory. However, the current version of Virtual *can* use some of the space above the top of ROM for virtual memory, provided the NuBus slot to which it's mapped isn't occupied.

To provide this feature, Virtual had to begin trapping requests for memory both from applications and from the operating system. Many programs, including MultiFinder, expect to allocate memory in contiguous chunks, and the space reserved for the ROM gets in the way. Virtual gets around this by trapping memory requests that would overlap the ROM—and turning the ROM into a nonrelocatable heap block.

If you're willing to put more than 8MB of physical RAM into your machine, Virtual can also do another neat trick. Some manufacturers have created NuBus memory cards that add memory in the NuBus slot space; alas, because the Macintosh Memory Manager wasn't designed to use this space, the RAM can be used only as a RAM disk or disk cache. Virtual, however, can let the system use this physical RAM in the same way it can put virtual memory at those addresses.

Your video card and the motherboard will always take up 2MB of the high memory space, and the ROMs currently consume only 128K of their 1MB, so you're left with 14.75MB of address space for real and virtual memory in 24-bit mode. (You always get this much on an SE 30, because you can't add cards to it.) But in 32-bit mode—now available to device drivers and usable by all applications under System 7—the memory space explodes to 4GB.

Figure 5b shows the frontiers of the Mac's brave new 32-bit world. A whopping 1GB of space is reserved for system RAM; the remaining space contains the ROM and peripheral cards (and, presumably, more physical or virtual memory). Each NuBus slot gets 256MB of the 4GB address space. But will the Macintosh ROMs work in this new configuration? A quick look at the latest ROMs shows that they will. Apple has cleverly set things up in anticipation of this scheme; all the internal jumps go to addresses such as 4080XXXXh. The "40" in the high byte, ignored in 24-bit mode, will direct the CPU to the correct address in 32-bit mode.

VIRTUAL VS. MAC VM

Frustrated Apple developers sometimes protest that as soon as a company markets a brilliant product for an Apple computer, Apple steps in and attempts to take over the market. In this case, their complaints may be justified: Mac VM, Apple's virtual memory manager for System 7, does exactly the same thing that Virtual does. The biggest difference is that Apple, which has control of the standard and the source code to the system ROMs, has been able to optimize System 7 and Mac VM to work together. New system routines let applications fix blocks permanently in physical memory; this makes it easier to ensure that a critical piece of code does not get swapped out at a bad time. Mac VM still can't do much to prevent Mac programs from corrupting one another's code or data, however; the current architecture of the Mac ROMs (and of many applications) will probably make memory protection impossible anytime in the near future.

There's enough to learn about Macintosh memory management to fill a chapter or two of a good-sized book. If you'd like to delve deeper, you can investigate some of the references in the bibliography. PJ

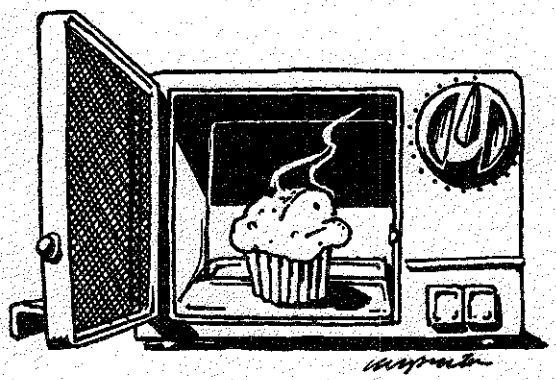
BIBLIOGRAPHY

- Apple Computer (1985). *Inside Macintosh, Volume II*. Reading, MA: Addison-Wesley.
- Shebanow, Andrew (1989). "Technical Note #212: The Joy of Being 32-Bit Clean." Cupertino, CA: Apple Computer.
- Chernicoff, Stephen (1988). *Macintosh Revealed, Volume 1: Unlocking the Toolbox*. Indianapolis, IN: Hayden Books.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DESQview-aware programs

Enhance BASIC with assembly language modules for DESQview's multitasking



By Murray L. Lesser

Being somewhat multilingual as far as programming languages are concerned, I try to use whichever language is best suited to the application at hand. I can't be bothered to learn the idiosyncrasies of each compiler vendor's "development environment," so I use Quarterdeck's DESQview 386 as a universal programming platform. Using this platform allows me to run lengthy jobs in the background while I write and test programs in the foreground.

My home-grown text-processing utilities are written in compiled Microsoft BASIC enhanced with assembled procedures. Since BC version 4.0, compiled BASIC programs write text-mode display output directly to the display-screen memory when they are linked only to the libraries furnished by Microsoft. Although DESQview 386 can multitask such programs, "plain" DESQview can run programs that write directly to display-screen memory only in a full-screen, foreground window.

I have outsmarted this deficiency by adding a few assembled modules to my BASIC private library to allow me to write "DESQview-aware" programs in BASIC. DESQview-aware programs can run directly under DOS, but they perform even better when running under DESQview. For example, my DESQview-aware BASIC utilities will run in small windows—or even in the background—on my 640K, diskette-only, 80C88-powered laptop, letting me take advantage of the DESQview user-interface capabilities.

In what follows, I present two assembled private-library modules. One is a function that returns a long-integer value corresponding to the DOS clock time. The second module makes command calls to the DESQview Application Program Interface (DVAPI) if the program is running under DESQview: one linkable procedure allows compiled BASIC programs to break out of their time slice while waiting in a polling loop; the other forces BASIC to write to DESQview's "shadow" display memory segment instead of directly to the screen memory. The use of these three procedures is illustrated by a short BASIC "alarm clock" program.

TIME DELAYS

DESQview is a preemptive multitasking environment that runs under DOS. Each active task is given its time slice in turn. Time slices are measured in BIOS clock ticks; typical values for an 80386-powered machine are nine ticks for the foreground task, followed by two ticks for each active background task. DESQview switches to the next task in the sequence when the current time slice is ended, when DESQview notes that the current task is waiting for a keystroke entry, or if the task executes a DVAPI PAUSE command. This last technique is the only way DESQview can "know" that it is just waiting around during a polling loop.

Programs assuming continuous control of all system resources do not operate correctly under DESQview. The most common unusable programs are those that measure time delays by counting BIOS clock tick interrupts. Acceptable time delays are measured by direct comparison with the DOS clock time (which is precise to 10 milliseconds) or with the underlying BIOS timer count (accurate to about 55 milliseconds).

In general, using a language library's "time delay" function in a program running under DESQview is a poor use of hardware resources: a well-designed library delay function is a polling loop that will most likely spend its entire time slice waiting for nothing to happen. SLEEP, the delay function introduced in BC 4.5, also contains a built-in booby trap: the delay is ended by any keystroke, so switching a task from foreground execution to background during a SLEEP delay ends the delay.

The traditional BASIC function for building time-delay loops (or for measuring time intervals) is TIMER. TIMER

Murray L. Lesser is a retired IBM engineer. He has written three books about enhancing various Microsoft BASIC compilers with assembled modules. The most recent, Advanced QuickBASIC 4.0, *was published by Bantam.*

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
        TITLE   Time in Hundredth-seconds
        .MODEL  MEDIUM
;                               HUNDSECS.ASM
; HUNDSECS [DECLARE FUNCTION HUNDSECS& ()]: an assembled function for com-
; piled BASIC programs that returns the system time of day in hundredths of
; seconds as a long integer. It serves the same purpose as the intrinsic
; BASIC function "TIMER" but doesn't require using floating-point arithmetic.
; Written by M. L. Lesser, December 25, 1990.
; Assembled with Microsoft MASM, version 5.1

.CODE
        PUBLIC  HUNDSECS
HUNDSECS PROC
        PUSH    BP
        MOV     AH,2CH          ;DOS "Get time" function
        INT     21H
        MOV     AL,60
        MUL     CH              ;Hours -> minutes in AX
        XOR     CH,CH
        ADD     AX,CX           ; + Minutes
        XCHG    DX,BX           ;Put Seconds and Hundreds in BX
        MOV     CX,6000         ;Hours + minutes -> seconds * 100
        MUL     CX              ;  in DX:AX
        XCHG    AX,CX           ;Now in DX:CX
        MOV     AL,100
        MUL     BH              ;Seconds * 100 in AX
        XOR     BH,BH
        ADD     AX,BX           ; + Hunds
        ADD     AX,CX           ;Low order of final value
        ADC     DX,0            ;High order
        POP     BP
        RET
HUNDSECS ENDP
        END
```

LISTING 2

```
        TITLE   DESQview Aware Procedures
        .MODEL  MEDIUM
;                               DVAWARE.ASM
; Two assembled subroutines to allow compiled BASIC programs to be "DESQ-
; view aware." These routines have no effect when the program is running
; directly under DOS.  DVBREAK [DECLARE SUB DVBREAK ()] stops the time
; slice if the program is running under DESQview. DVSHADOW [DECLARE SUB
; DVSHADOW ()] allows programs compiled with BC 4.0, or later, to run in
; the background or in small windows under "plain" DESQview. The CALL
; DVSHADOW statement must be executed before any PRINT statement in the program.
;   WARNING:  BASIC programs using either of these procedures must be com-
; piled with switch "/O" for standalone use.
; Written by M. L. Lesser, August 7, 1990
; Assembled with Microsoft MASM, version 5.1

IF1
        INCLUDE DVAPI.INC               ;From DVAPI Reference Library
ENDIF

.DATA
        EXTRN   B$VideoBase: WORD       ;Base of (text) screen memory

.CODE

DVPRES  PROC NEAR               ;Local (non-public) subroutine to check
        @CALL   DVPRESENT       ;  for DESQview presence.
        OR      AX,AX           ;Will be zero if not running under DV
        RET
DVPRES  ENDP

        PUBLIC  DVBREAK
DVBREAK PROC
        PUSH    BP
        CALL    DVPRES
        JZ      DONE                    ;If not under DV, quit
        @CALL   PAUSE                   ;Else drop time slice
        JMP  SHORT  DONE
DVBREAK ENDP

        PUBLIC  DVSHADOW
DVSHADOW PROC
        PUSH    BP
        CALL    DVPRES
        JZ      DONE                    ;If not under DV, quit
        @CALL   SHADOW                  ;Else tell DV to shadow output
        MOV     B$VideoBase,ES
DONE:   POP     BP
        RET
DVSHADOW ENDP
        END
```

LISTING 3

```
'                               ALARM.BAS
'    ALARM allows a pseudo alarm clock to be set and rings system bell when
' time is reached.  Alarm is designed to operate in the background or in a
' small window under DESQview. WARNING: ALARM must be compiled for
' standalone ("/O" switch) for DVSHADOW to operate.
' Written by M. L. Lesser, 12/25/90
' Compiled with Microsoft BC 7.1, switch "/O". Linked to HUNDSECS, DVAWARE,
' and the BASIC 7 stub file NOFLTIN with switches "/E/NOE"


   DEFINT H,M                   'Integer variables: HERE, HOURS, MINUTES
   DEFLNG N,A                   'Long variables:  NOW, ALARM
   DECLARE FUNCTION HUNDSECS& ()
   DECLARE SUB DVBREAK ()
   DECLARE SUB DVSHADOW ()

   DVSHADOW                     'No effect if not running under DV
   CLS
RESTART:                'Restart point for invalid input alarm time
   LET HERE = CSRLIN
   GOSUB SHOWTIME               'Use GOSUB to save space and time
   PRINT
   INPUT "Enter alarm-set time (hh,mm):  ", HOURS, MINUTES
   IF HOURS < 0 OR HOURS > 23 OR MINUTES < 0 OR MINUTES > 59 THEN
      PRINT  "Invalid set-alarm time.  Enter again."
      GOTO RESTART
   END IF
   LET ALARM = 360000& * HOURS + 6000& * MINUTES
' ALARM is the alarm setting in hundredths of seconds. If alarm is set
'  before current time, assume alarm is intended to be pm; add
'  4,320,000 hundredth-seconds (12 hours) to setting time.  If the
'  result is now greater than 8,640,000 hundredth-seconds, or is still
'  less than current time, assume an attempt was made to set the alarm
'  for tomorrow and terminate the program.
   IF ALARM <= HUNDSECS THEN
      LET ALARM = ALARM + 4320000&
      IF ALARM >= 8640000& OR ALARM <= HUNDSECS THEN
         PRINT "  Cannot set alarm for tomorrow"
         PRINT "Press any key to terminate program"
         WHILE LEN(INKEY$) = 0: WEND
         END
      ELSE
         LET HOURS = HOURS + 12
      END IF
   END IF
   PRINT "  Alarm is set for" STR$(HOURS) ":";
   IF LEN(STR$(MINUTES)) = 2 THEN PRINT "0";
   PRINT LTRIM$(STR$(MINUTES))
   LET HERE = CSRLIN
   WHILE HUNDSECS < ALARM               'Wait for the alarm to go off
      GOSUB SHOWTIME                    '  showing current time
      IF INKEY$ = CHR$(3) THEN END      '   but allow Ctrl-C escape
      DVBREAK                           'Exit time slice if under DV
   WEND
   PRINT: PRINT "Press any key to turn off bell and terminate program."
   IF CSRLIN = 24 THEN LET HERE = 22
AROUND:                         'Start of GOTO loop ringing alarm bell
   LET NOW = HUNDSECS + 100&            'Actually, one second from now!
   BEEP                                 'Ring bell at
   WHILE HUNDSECS < NOW                 '  one-second intervals
      DVBREAK
      GOSUB SHOWTIME                    '  showing time of day
   WEND
   IF LEN(INKEY$) = 0 THEN GOTO AROUND  '  until key is pressed
   PRINT
END

SHOWTIME:       'Local subroutine displaying time of day in place:
   IF HERE > 23 THEN
      PRINT: PRINT
      LET HERE = HERE - 2
   END IF
   LOCATE HERE, 3
   PRINT "It is now:  " TIME$;
RETURN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

suffers from the typical ills of computing in floating point: false precision indications, poor performance, and the need to include lengthy emulator code in the executable file. The assembled function HUNDSECS.ASM (Listing 1) returns the DOS clock time, in hundredths of a second since midnight, as a long integer. Text-only programs properly written to use HUNDSECS instead of TIMER can be compiled and linked to omit the floating-point emulator, thereby saving up to 13K of code space.

THE SHADOW KNOWS

The module DVAWARE.ASM (Listing 2) contains three procedures, only two of which have public entry points. (Omitting the language name from the MODEL directive prevents MASM 5.1 from forcing all PROC names to be PUBLIC.) DVPRES, a local procedure, is called by each of the two public procedures—DVBREAK and DVSHADOW—to determine whether or not the program is running under DESQview. If it isn't, neither public procedure has any effect.

Listing 2 contains macros defined in the DESQview DVAPI.INC header file. DVAPI.INC is distributed on diskette with the DESQview API Reference and with any DESQview API (DVAPI) language library. The assembly language expansions of the three @CALL macros I use here are also shown in Appendix J of the DESQview version 2 manual.

The DVBREAK subroutine is used to break out of an active time slice, and it should be included in every polling loop unless you expect the polling period to be less than the minimum time slice.

The DVSHADOW subroutine allows compiled BASIC programs to run in small windows, or in the background, when the programs are running under non-386 DESQview. DVSHADOW then loads the segment address of DESQview's "shadow" display memory into the BASIC global variable **B$VideoBase.**

B$VideoBase exists as a global variable only in programs compiled with the "/O" switch for stand-alone execution. Attempting to link this module into a BASIC program that has been compiled to use BASIC's Run-Time Module produces an "unresolved external" error message. I assume that the module would cause some sort of runtime failure if it were linked into a "Quick" library and used with QBX (or QB), although I have never tried it. Early experience with QuickBASIC demonstrated that many programs would run correctly when compiled with the command-line compiler but not under the "development environment." For other programs, the reverse was true.

Programs that call DVSHADOW (before executing any PRINT statement) should be installed under DESQview with "no" in answer to the "writes directly to screen" and "virtualize text" questions.

Even if you have the DESQview API BASIC Library, you should use the assembled procedures shown here for writing DESQview-aware programs, rather than trying to use the calls in that library. The BASIC Library API calls are useful only for DESQview-specific BASIC programs. Even so, the library does not support the API SHADOW call. You have to add an assembled DVSHADOW procedure if you want your BASIC DESQview-specific programs to run in the background under plain DESQview without using QB45LOAD.COM, which is the library-furnished loader program. (Note that QB45LOAD.COM works only with BASIC programs compiled with the "/O" switch.)

SOUND THE ALARM

Although ALARM.BAS (Listing 3) is a fairly simple program, it serves to illustrate a few of the niceties of running a BASIC program under DESQview using minimal memory. ALARM was written to take into account idiosyncrasies of both the compiler and the DESQview environment. If compiled with BC 7.1 and linked as noted in the listing, ALARM.EXE will run in a DESQview window set for 21K "Memory size" and zero "System Memory" and "Script Buffer" sizes. If compiled with BC 4.5, the program requires 34K of memory.

ALARM requests an alarm-set time from the user. If a valid time is set, ALARM goes into a timing loop (interrupted on each pass by DVBREAK) until the set time is reached. After that point, ALARM goes into a second timing loop, ringing the PC's bell each second until ALARM's window is switched into the foreground and a key is pressed.

The Microsoft BASIC compilers are

Commenting Disassembler!

SOURCER™ 486

- **SEE HOW PROGRAMS WORK**
- **EASILY MODIFY PROGRAMS**

SOURCER™ creates commented source code and listings from executable files or memory, suitable for reassembly. Sourcer helps clarify the inner workings of programs, with detailed in-line comments on interrupts, subfunctions, I/O port functions, and much more. It offers complete support for all 8088/87 to 80486 instructions and V20/V30.

Sourcer provides the most comprehensive automatic analysis available to accurately separate code and data. It determines data types, uses descriptive labels for BIOS and PSP data, and links data items across multiple segments.

Professionals consistently chose Sourcer because of it's ability to achieve far superior results with the least effort.

"Sourcer is the best disassembler we've ever seen." PC Magazine

BIOS SOURCE

for PS/2, AT, XT, PC and Clones

- **CHANGE AND ADD FEATURES**
- **CLARIFY INTERFACES**

The BIOS Pre-Processor™ augments Sourcer by allowing you to obtain commented listings for any BIOS ROM in your machine. Now you can understand how your specific BIOS works. Adds over 75K of comments specific to your BIOS. Tracks and identifies multiple interrupt branches with special labeling such as "int_10_video." Fully automatic operation.

ASM ProPak™

Combine our advanced assembly language tools and save up to $120! All ASM ProPaks include Sourcer & BIOS Pre-Processor, Unpacker,™ to unpack packed EXE files, and View-It,™ our high speed file viewer which can display 132 columns. ASM ProPak II includes ASMtool™ 486 to flowchart 8088 to 80486 source code. ASM ProPak III adds ASM Checker,™ the only assembly source code bug finder.

Sourcer 486–Disassembler	$129.95
BIOS Pre-Processor	49.95
Sourcer w/BIOS–(save $10)	169.95
Unpacker–Unpacks packed files	39.95
View-It–View 132 column files	69.95
ASM ProPak I–All Above (save $40)	249.80
ASMtool 486–Automatic flowcharter	199.95
ASM ProPak II–All Above (save $90)	399.75
ASM Checker–Finds source code bugs	179.95
ASM ProPak III–All Above (Save $120)	549.70
Memory Commander®–Get up to 900K of contiguous DOS memory	99.95

Shipping and Handling: USA $6; Canada/Mexico $10; Other $18. CA Residents add sales tax. © 1991 Visa/MasterCard/COD accepted.

30-DAY MONEY-BACK GUARANTEE

1-800-648-8266

V Communications Incorporated
4320 Stevens Creek Blvd., Suite 275-PJI
San Jose, CA 95129 FAX: 408-296-4441
(408) 296-4224

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

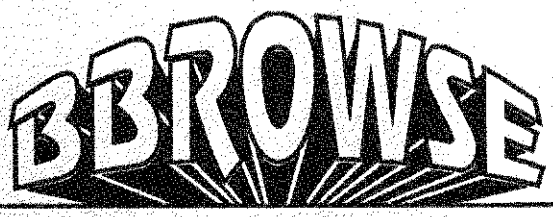
not as smart as they should be. Even though a program appears to have eliminated all statements that use floating-point arithmetic, BC still asks the linker for the emulator if it finds any statement containing an INPUT, READ, or VAL function; the compiler doesn't notice that the same statement also contains the typed variable to which the requested value is to be assigned. The stub file NOFLTIN.OBJ was introduced with BC 7.0. If NOFLTIN is linked with a program containing INPUT or READ statements, it replaces the full emulator with a dummy emulator module, saving about 13K of code space. A bug in the BC 7.0 VAL compilation was worsened in BC 7.1: if the source code contains a VAL statement, NOFLTIN does not prevent the full emulator from being linked into the executable file.

The workaround in ALARM is to avoid using VAL. Instead of parsing a single string input *(hh:mm)* variable to obtain the alarm-set time, the program requests a pair of integer variables *(hh,mm)* that are separated by a keyed-in comma. The neophyte user will receive several unfriendly "Redo from start" error messages until the lesson is learned.

When ALARM is added to DESQview's repertoire, the options are set to block DESQview's "Close window" command and to disallow ALARM's being swapped out of active memory. These options prevent users from inadvertently turning off the alarm before its time, and they assure that ALARM will keep running in the background as long as its window is open. Because there has to be some way to close ALARM's window, the "Close on exit" option is also set.

Using the "Close on exit" option allows ALARM.EXE to be loaded by DESQview's own loader, rather than by a copy of COMMAND.COM, thereby reducing the minimum program memory size. Directly loaded programs should not contain any automatic error terminations, because any message explaining the termination will be lost when the window vanishes. This is illustrated in ALARM by the forced termination of any attempt to set the alarm for the next day. Although the termination cannot be prevented, it is not effective until the "error" is acknowledged with a keystroke. ALARM can be terminated before its time by switching its window to the foreground and pressing Control-C.

ALARM was written originally to be a test-bed for studying DESQview-aware programming in BASIC, and it has been enhanced several times to add function while reducing its minimum program memory size. In spite of its being only an exercise, I find it a useful utility to prevent my becoming so immersed in the computational muse that I forget an appointment. My reminder to set the alarm is a DESQview "startup" script (see the DESQview version 2 manual, Chapter 5), which opens the ALARM window for me at the beginning of every DESQview session. PJ



Extends BRIEF to include a multi-language browser

- ***Languages Supported:*** Microsoft C 6.0, MASM 6.0, BASIC 7.1, COBOL 4.0, FORTRAN 5.1 and later versions.
- ***Project-wide Symbol Usage:*** Symbol Details shows how a particular symbol is used throughout your multi-file project.
- ***Hypertext Editing:*** From within a report, a single keypress brings up your source file for editing, with your cursor at the position of the definition or reference.
- ***Find Dead Wood:*** The Unused Symbols Report finds abandoned functions and variables.
- ***Generate Documentation:*** All reports can be printed to satisfy your documentation needs.
- ***NOT A TSR:*** BBROWSE is an extension of BRIEF, it doesn't take up your valuable memory space.
- ***Accuracy:*** Microsoft generates the database that BBROWSE uses. You get an exact representation of your code, NOT just an *interpretation* from a generic source code scanner.

BBROWSE requires a Microsoft Professional Language and BRIEF3.0+

BBROWSE: *only* **$89**

Call: 1-303-465-5277

FREE DEMO DISK AVAILABLE!

THE SOFTWARE ANNEX

3143 W. 12th Avenue Court, Broomfield, CO 80020

PJ Code is available on the following bulletin board services:

The Buffer Board
538 Cardinal Way, Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from the Sept/Oct 1987 issue through the current issue.

PCS-49 on CompuServe
Section 10, sponsored by: MISOSYS, Inc.
P.O. Box 239, Sterling, VA 22170
(Voice) 703-450-4181

The Silicon Zone
4419 W. Hillsboro Blvd., Suite 290, Coconut Creek, FL 33073
(Data) 305-360-9376

Generation Five Inc.
12061 Tech Road, Silver Spring, MD 20904-9998
(Voice) 301-680-3600 **(Data) 301-680-9300**
Accessible via PC Pursuit, Washington, D.C. From Files submenu, select "P" for Personal Downloads.

The Programmer's Corner
5430F Lynx Ln. #224, Columbia, MD 21044
(Data) 301-596-1180, 621-4636, or 995-3744

A.C.M. Phoenix Chapter
6408 East Palm Lane, Scottsdale, AZ 85257-2563
(Voice) 602-990-3102 **(Data) 602-970-0474**

Gilmore Systems of Southern California
1285 Falling Star Avenue, Thousand Oaks, CA 91362
(Voice) 818-706-9800 **(Data) 818-706-9805**

Mnematics Videotex
P.O. Box 19, Sparkill, NY 10976-0019
(Voice) 800-322-3633 (Voice) 914-365-0184 in N.Y.
Mention PJ when you call.

Circle 29 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CONTEST

How I got rich and famous reading Programmer's Journal

by Programmer Bob

t was a bright and sunny day. There I was, relaxing with my 50-MHz 486 PC laptop beside my Olympic-sized swimming pool, reading my favorite magazine, **Programmer's Journal.** By the way, I'm Bob, the tanned and sophisticated computer programmer. As I devoured the code-intensive pages, the enlightening articles, and a platter of chilled crudités, I imagined how perfect life would be if only I had a **650-MB Sony Rewritable Optical Disk Drive with media.** Suddenly, I saw a photo of the drive along with the words **Contest** and **Win.** The marvelous folks at **Programmer's Journal** were giving me a chance to own the hardware of my dreams. All I had to do was send them a story about one of the many occasions when some tip or article in **PJ** had saved me time and money. "I'm really too busy for this," I thought as I sipped my iced tea. "But everyone loves hearing about the time when I was designing the [illegible] killer applicat[illegible]

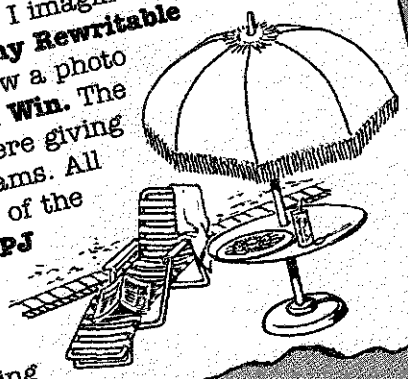
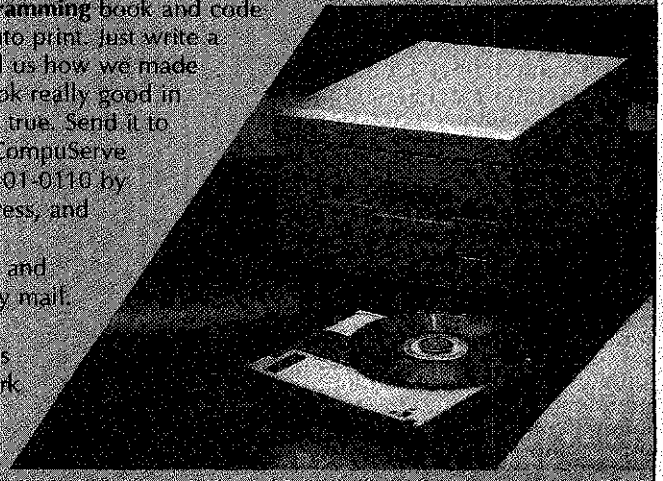


ILLUSTRATION BY JIM O'MORROW

Bob got rich and famous reading **PJ.** And you can, too. Because EVERYONE who enters wins a special gift. Grand prize is **Sony's revolutionary 650-MB Optical Drive,** with interface kit and media (a $6,000 value). Five people will win our second prize, a coveted **Edsun Labs CEG/DAC, VGA replacement chip and SDK.** The ten third-place winners get Michael Abrash's **Power Graphics Programming** book and code disk. If you're a winner, you and your story will make it into print. Just write a page or so about some article or tip you've read in **PJ.** Tell us how we made your life easier, made you richer, or how we made you look really good in front of your friends. It has to be original. And it has to be true. Send it to **PJ's Amazing Stories Contest** via FAX (503) 746-0071 or CompuServe 76676,427, or mail it to P.O. Box 70167, Eugene, OR 97401-0110 by **October 1, 1991.** Don't forget to include your name, address, and phone number.

We'll announce the winners in our Jan/Feb 1992 issue and print their winning stories. Winners will also be notified by mail. Oakley Publishing will be the sole judge of which entries qualify and are valid, and its decisions are final. All entries become the property of Oakley Publishing. And if you work for Oakley, you're not eligible. By the way, our judges are a trifle eccentric. The stranger the tale, the better. So go ahead . . . make their fans fade.

WIN



650-MB Sony Rewritable Magneto Optical Drive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Windows property lists

Use Windows 3 features to simplify linked-list management

By William S. Hall

Based on what you read about Windows 3.0 enhanced mode, you might expect that memory is no longer an issue in Windows programming. Not so! DOS may have its 640K limit, but Windows also has some significant—albeit different—limits when it comes to managing memory. This article addresses some of these memory issues and describes some solutions that are based on a Windows network-management application we were developing.

One of the first tasks we faced in developing a Windows program to manage networks was to draw a logical map of each network segment discovered by the hardware device. Each map was to show the various nodes on the segment (PC, bridge, router, etc.) and was to be drawn in a separate multiple document interface (MDI) child window of the main program.

With an unpredictable number of nodes, the map could be described easily using a doubly linked list in which each list element contained descriptive information about its corresponding network node. At the beginning of the project, we simply created a list for each segment in the program's local heap using the rich set of heap-management functions available in Windows. As the program grew and various program features became operational, Windows felt the pinch in its one 64K data segment. It was simple to work around this limit by moving the list to the global heap, because a number of the local and global heap functions mimic each other. For each network segment, memory was allocated from the global heap.

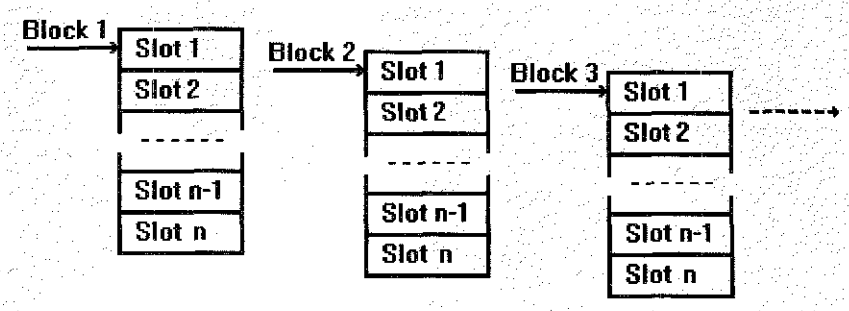
Then, one day, we connected the network hardware to a large government network. The Windows program ground to halt: it had run out of selectors. At the time, we were naively using one selector per node—but the target network had segments with thousands of nodes. With a maximum of 8,192 selectors available in Windows, it was time to find an alternative approach to managing the linked list.

The solution was to manage the memory in uniformly sized blocks divided up into slots of equal size (Figure 1). Each slot would contain the information that previously had been contained in one element of the list. By using just one selector for each block and many slots per block, the demand for selectors could be greatly reduced. Of course, such a scheme would require a certain amount of overhead for storing the current block and slot numbers, as well as pointers to each block. And each set of data needed to be associated with the window supporting the network map.

PROPERTY LISTS RIDE AGAIN

Thus, the block memory-management system suggested the use of a *property list*. A property list allows you to associate a number of arbitrary 16-bit quantities with a given window. You need two things: a property name (a character

FIGURE 1: Diagram of the memory-management system



William S. Hall is a senior engineer at Novell, in San Jose, California. He specializes in Windows software.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

string) and a handle to a window (to place the data item on the window's property list). You can retrieve, remove, or enumerate this data at any time. By using the property list, you can simplify information storage and avoid building complex data structures.

In what follows, I will describe the various Windows property list functions and illustrate them with a Windows program that contains the elements of the memory-management scheme described above. The program also shows how to simplify the display of information by using a predefined control class (in this case a listbox) in your main window.

TOGETHER AT LAST

There are two ways to store data that is unique to a given window. If you know the size of the data, you can use the **cbWinExtra** field in the WNDCLASS data structure. When the Windows class is registered, just declare the amount of extra data space you want. If you need to tie an unknown number of 16-bit entities to a given window, however, a property list becomes convenient.

Property lists are storage areas that contain handles for data associated with a window. Property lists are especially useful for working with predefined Windows classes such as EDIT, LISTBOX, and DIALOG. Because you have no control over the allocation of extra bytes, they cannot be assigned in the class structure, but you can store information conveniently in a property list as long as the window exists. Of course, each property requires a unique string, but these are easily manufactured by concatenating a single string with a unique integer value that has been converted to a string.

PROPERTY LIST FUNCTIONS

Four functions are available for manipulating the property list. **SetProp** adds entries to the list. Each entry contains a unique text string and a data handle; the string identifies the handle, and the handle describes the data. Instead of strings, you can use atoms (integers that uniquely identify strings) to describe properties. The atom must first be created by the **AddAtom** (or **GlobalAddAtom**) function. When you call **SetProp,** you then pass it the atom rather than the string.

Once an item is in the property list, any application can access the data using **GetProp** if the window handle and the string (or atom) are known. Information can be shared, but the usual caveats about using another program's handles apply: only global memory objects—such as GDI objects—can be used by other applications. If a property list contains local or global memory handles or resource handles, they belong exclusively to the program that created them.

EnumProps enumerates the contents of a property list. This function requires that the application declare a callback function to receive the data handle and string of each entry in the list. The callback function can perform any task.

Properties are removed from the list using **RemoveProp.** When a window is destroyed or when the program terminates, it is especially important to remove all the properties that were added by the application.

Although the data handles in a property list always belong to the application that created them, the property list itself belongs to Windows. The property list is allocated in the local heap of the USER library (USER.EXE); this space is also used for other data related to each window in the system. Because the USER data segment—and hence its local heap—cannot exceed 64K, excessive use of property lists can deplete Windows' resources. For a discussion of resources (extracted from the Windows SDK), see the "Windows Resource Limits" box on p. 54. You can view the current state of resources in the Program Manager's "About" box (Figure 2).

The Presentation Manager (PM) SDK contains a programmer's note that can give you insight into how property lists work. PM has no property lists, but you can achieve the equivalent function by creating a linked list in memory and placing a pointer to it in the Windows extra-bytes area. Of course, you must construct the corresponding property-management functions as well.

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2: The Program Manager's "About" box

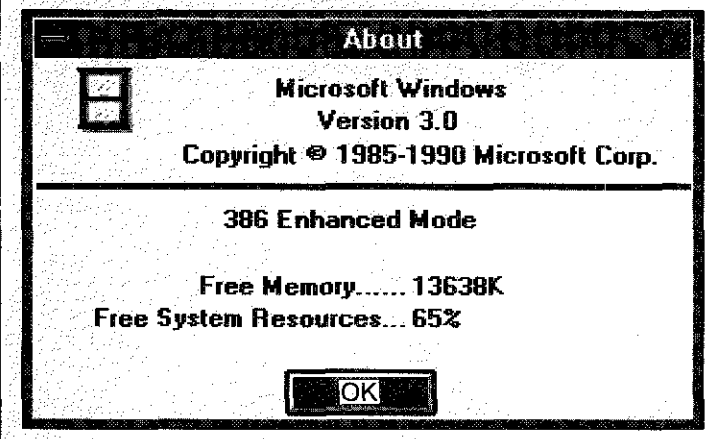
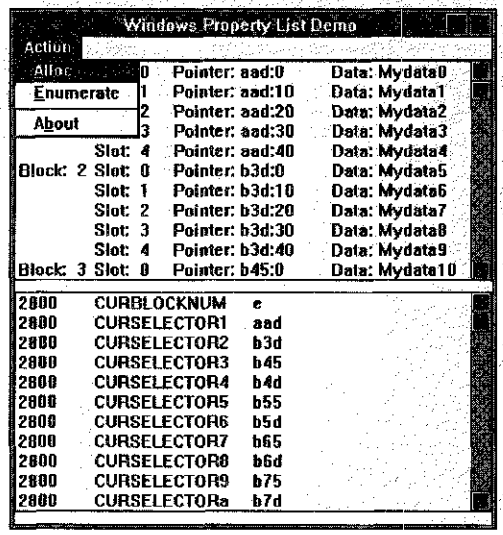


FIGURE 3: Property list enumeration


Windows Property List Demo
Action
Alloc
Enumerate
About
0 Pointer: aad:0 Data: Mydata0
1 Pointer: aad:10 Data: Mydata1
2 Pointer: aad:20 Data: Mydata2
3 Pointer: aad:30 Data: Mydata3
Slot: 4 Pointer: aad:40 Data: Mydata4
Block: 2 Slot: 0 Pointer: b3d:0 Data: Mydata5
Slot: 1 Pointer: b3d:10 Data: Mydata6
Slot: 2 Pointer: b3d:20 Data: Mydata7
Slot: 3 Pointer: b3d:30 Data: Mydata8
Slot: 4 Pointer: b3d:40 Data: Mydata9
Block: 3 Slot: 0 Pointer: b45:0 Data: Mydata10
2800 CURBLOCKNUM e
2800 CURSELECTOR1 aad
2800 CURSELECTOR2 b3d
2800 CURSELECTOR3 b45
2800 CURSELECTOR4 b4d
2800 CURSELECTOR5 b55
2800 CURSELECTOR6 b5d
2800 CURSELECTOR7 b65
2800 CURSELECTOR8 b6d
2800 CURSELECTOR9 b75
2800 CURSELECTORa b7d

HEAPING BIG MEMORY

WALLOC.EXE uses the property list functions to implement the memory-management scheme described above. The goal is to provide a long pointer to a fixed-size chunk of memory in the global heap on each allocation request. The memory can be used in any way. In its original implementation in the network-management program, the block is an element in a linked list containing data and pointers to the next and previous members. Generally, the list only grows, and it exists for the life of the window. Nodes are occasionally moved but rarely removed. Hence, there is no need for memory deallocation (or garbage collection and compacting) except when the window that manages the list is itself destroyed.

The scheme uses property list strings to represent the current block number, the current slot within the block, and the block's selector (or, if in real mode, segment) value. There is also a constant value (not placed on the property list), which is the number of memory slots in each memory block.

On each allocation request the following sequence occurs. If a new block is to be allocated, a call is made to Windows via **GlobalAlloc.** The allocation size is the requested size multiplied by the number of memory slots in each block. (If this latter value is *n*, then the number of selectors needed per chunk of memory is reduced by this factor.) If the allocation succeeds, the required memory will be used for a long time, so the handle that is returned is wired by using **GlobalWire.**

Then the three properties on the list are updated. The first property is the current block number, which is incremented by 1. The second is the slot number, which is reset to 0 because this is the first slot in a new block. Finally, the selector (or segment) returned by **GlobalWire** is placed on the property list. The string used for this property is a concatenation of a constant string and the block number. The allocation function returns with the long pointer obtained from **GlobalWire**, or with NULL if the allocation fails or the property list is full.

If the slot number has not exceeded the maximum number of slots per block, then the property containing the slot number is incremented by 1. The current selector on the property list is read, and then an offset value is constructed by multiplying the current slot number by the size of the memory allocation. The selector and offset thus calculated are returned to the caller as a long pointer.

As mentioned above, there is no requirement for deallocation of memory during the life of the window. When it is destroyed, however, each added property is removed, and the corresponding memory block is unwired and freed.

After running the program, you'll see three commands listed under the Action menu: **Alloc, Enumerate,** and the ubiquitous **About** box. Each time you activate **Alloc,** another chunk of memory is assigned. Information about its allocation block, slot number, pointer value, and the stored data is added to a listbox that fills the top half of the window's client area.

You can enumerate the current property list at any time using the **Enumerate** command. A listbox on the lower part of the screen displays the properties (see Figure 3).

Accelerator keys make it easy to perform a series of allocations: just hold down the "ALT" and "A" keys. To view the current property list, press the keys "ALT+A+E."

As mentioned, properties end up in USER's heap. You can run the HeapWalker program from the Windows SDK and look for USER's data segment in HeapWalker's window. Double-click on that line to produce a dump of the segment. Now scroll through until you find the expected property strings. Figure 4 shows a partial dump of the heap and the property strings during a run of WALLOC.

THE INITIALIZATION UNIT

A partial listing of the program accompanies this article, and the complete source is available on the *PJ* disk for this issue.

WALLOC is written in the medium model with three code segments. The initialization segment (Listing 2) contains only the routines needed at start-up.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All is standard except for the creation of two child windows of the listbox class in **WndCreate.** In place of the menu handle, a unique identification number is assigned to each listbox. That way, each child can be treated like a control in a dialog box.

Note that if these child windows cannot be created, then the main program is terminated by posting a WM_SYSCOMMAND message to itself with word parameter SC_CLOSE (instead of by returning FALSE from **InitProgram).** Once the main window has been created, the main message loop begins to execute. If **InitProgram** were to return FALSE at this point, a general protection fault would result.

THE MAIN WINDOW MODULE

The main window procedure is shown in Listing 3. Note the response to WM_SIZE: the two child listboxes are repositioned and sized each time the main window is resized.

This module also contains the menu responses under WM_COMMAND. When the **Alloc** menu command is selected, another chunk of data is allocated; a unique string is copied there and the upper list box is updated.

When **Enumerate** is executed, the properties that have been added to the main window are obtained and placed in the lower listbox. Note the call to reset the redraw flag before the listbox is emptied and filled again; only when

FIGURE 4: Partial dump of the heap and property strings

```
                0007ADA0 06EE  28448 1    User      Data
47B0 0B 43 55 52 42 4C 4F 43 4B 4E 55 4D 00 00 00 00   CURBLOCKNUM
47C0 A9 47 D4 47 00 00 01 00 0A 43 55 52 53 4C 4F 54  ©GÔG      CURSLOT
47D0 4E 55 4D 00 C1 47 EC 47 00 00 01 00 0C 43 55 52  NUM.ÁGìG.    .CUR
47E0 53 45 4C 45 43 54 4F 52 31 00 00 00 D5 47 04 48  SELECTOR1   ÕG.H
47F0 00 00 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  .    .CURSELECTOR
4800 32 00 00 00 ED 47 1C 48 0C 2A 01 00 0C 43 55 52  2   íG.H.*    .CUR
4810 53 45 4C 45 43 54 4F 52 33 00 00 00 05 48 34 48  SELECTOR3    H4H
4820 00 00 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  .    .CURSELECTOR
4830 34 00 00 00 1D 48 4C 48 00 00 01 00 0C 43 55 52  4    HLH    .CUR
4840 53 45 4C 45 43 54 4F 52 35 00 00 00 35 48 64 48  SELECTOR5    5HdH
4850 00 00 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  .    .CURSELECTOR
4860 36 00 00 00 4D 48 7C 48 48 2A 01 00 0C 43 55 52  6   MH|HH*    .CUR
4870 53 45 4C 45 43 54 4F 52 37 00 00 00 65 48 94 48  SELECTOR7    eH.H
4880 00 00 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  .    .CURSELECTOR
4890 38 00 00 00 7D 48 AC 48 00 00 01 00 0C 43 55 52  8   }H¬H    .CUR
48A0 53 45 4C 45 43 54 4F 52 39 00 00 00 95 48 C4 48  SELECTOR9    HÄH
48B0 00 00 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  .    .CURSELECTOR
48C0 61 00 00 00 AD 48 DC 48 8C 4A 01 00 0C 43 55 52  a   ¬HÜH.J    .CUR
48D0 53 45 4C 45 43 54 4F 52 62 00 00 00 C5 48 F4 48  SELECTORb    ÅHôH
48E0 AC 47 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  ¬G    .CURSELECTOR
48F0 63 00 00 00 DD 48 0C 49 50 49 01 00 0C 43 55 52  c   ÝH.IPI    .CUR
4900 53 45 4C 45 43 54 4F 52 64 00 00 00 F5 48 24 49  SELECTORd    õH$I
4910 CC 45 01 00 0C 43 55 52 53 45 4C 45 43 54 4F 52  ÌE    .CURSELECTOR
4920 65 00 00 00 0D 49 4C 49 BC 27 4E 4B 54 D2 00 00  e    .ILI¼'NKTÒ..
```


Resource Converter

Translate your resource files from **PM to Windows,** or **Windows to PM.** Save weeks of trial and error converting them yourself. Get the job done right the first time with the **Resource Converter!**

Converts:

Dialog Templates
All Control types
Std. Control ID's
Menus & Tables

PM to Win converter now shipping for $225, Win to PM shipping soon.

Petrasoft
3665 East Bay Drive Suite 204-112
Largo, Florida 34641
Msg: 813-530-9481, FAX: 813-530-9264

All trademarks or tradenames used here-in are owned by their respective companies.

Circle **30** on Reader Service Card

Memory Problems Solved!

MS-DOS 'C' AND ASSEMBLER PROGRAMMERS - BREAK THE 640K LIMIT!

XMEM A complete virtual memory system! **$149**
- Manages expanded, extended, and can swap to disk.
- Eliminates memory fragmentation problems.
- Optionally emulates Windows & Macintosh memory managers.

EMPTY SHELL Run any program from your program! **$169**
- Shell to any program, leaves less than 10K resident!
- Saves and restores screen - text or graphics!
- Swaps your program to expanded memory, extended memory or disk.

MEMORY TOOLKIT Easy access to all memory! **$99**
- Read and write expanded or extended memory with simple file like calls.
- Create multi-megabyte arrays in expanded, extended, or on disk.
- Complete set of calls to EMS and XMS managers.

To Order Call:
1•800•755•7973
Visa / MasterCard / C.O.D.
International: (609) 587-7973
Fax: (609) 587-9412

Kandu Inc.
"We Can Do What Others Can't"

ALL OUR PRODUCTS FEATURE: • Source code • Tested with Microsoft, Turbo, & Zortech C • 30 Day Money-back Guarantee • No Royalties! • 100% Assembler • Free technical support

Circle **31** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Program FASTER with ...

CC-RIDER

or the new

C++ RIDER

POP-UP SOURCE BROWSERS FOR ANY EDITOR!

CROSS-REF Editing!
Instantly pop-up your symbol definitions with a hypertext hot-key inside any editor! The Standard Edition has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all uses of a symbol in your code as you edit!

Source Analysis!
Build a database of symbol information with our powerful ***source code analyzer*** which generates a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE and HOW all your symbols are used. And it uses expanded memory for ***lightning fast*** source code analysis!

QuickHelp, too!
CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.***

Works with ANSI/Microsoft C, plus AT&T, Glockenspiel, Zortech, Borland C++. Supports Microsoft C v6 extensions.

FREE DEMO !

Standard	Professional
$89	**$279**
DOS or OS/2 C or C++	DOS & OS/2 C & C++

Western Wares
(303) 327-4898
Box C Norwood, CO 81423

Circle 32 on Reader Service Card

VLIB

The User Interface Library for C

Create Fast and Compact Applications in C & C++

✦ Menus ✦ Forms ✦
✦ Editor ✦ Windows ✦
✦ Mouse Support ✦
✦ Pick Lists ✦
✦ Popup Messages ✦
✦ Dialog Boxes ✦
✦ Colors ✦ and more! ✦

VLIB is a library of over 300 C functions for building PC applications. VLIB produces the *smallest, fastest* programs of *any* C library!

For *Microsoft C* and *Quick C, Borland Turbo C* and *C++*. All memory models.

Free Demo Disk Available

Call (408) 984-2256
Pathfinder Associates
291 Madrone Avenue
Santa Clara, CA 95051
FAX (408) 244-5665
BBS (408) 246-0164

VLIB $ 149
libraries, manual, and source code.

VMAN $ 49
online hypertext manual for VLIB.

Circle 33 on Reader Service Card

the process is complete is the redraw flag set and the listbox repainted. Otherwise, the listbox would be redrawn every time it is emptied and every time another entry is added by the enumeration callback function. The result would be an unpleasant flashing that is painful to the eyes.

This module also contains a useful routine called **dbs.** Windows 3 gave us CodeView but took away a helpful debugging tool: writing formatted information to the debugging monitor using **printf. dbs** provides a handy replacement. It is easy to add to any Windows program, but you should include STDIO.H and STDARG.H in the routine where it is defined. **dbs** is prototyped in WALLOC.H.

Note that you will need Michael Geary's OX.SYS to see the **dbs** output on a secondary monitor. OX.SYS has been around for many years; a copy is included on the *PJ* disk. Install it with the line **device=ox.sys** in CONFIG.SYS.

Listing 4 shows the allocation and deallocation functions, explained above. Also included here are the property enumeration callback and the functions that update the listboxes. The listboxes are treated as controls using **SendDlgItemMessage** with their window numbers as the control IDs.

Why does **AllocMem** remove the current block and slot number properties each time they are changed? In fact, the call to **RemoveProp** is not necessary, but it improves the **EnumProp** display. Whenever the same property is set more than once, the "string" returned by the **EnumProp** callback seems to be an atom that is local to USER. You can see this peculiar behavior by commenting the three **RemoveProp** calls and rebuilding the program. Any attempt to retrieve the string by either **GetAtomName** or **GlobalGetAtomName** results in a general protection fault.

The **make** file (included on disk) is for use with the Microsoft 6.0 C compiler. The target, WALLOC.EXE, is organized so that it can be built independently by NMAKE.EXE (or NMK.EXE) to accommodate the separate phases of linking or resource attachment. Just substitute a double colon for the single colon that separates the final target and its dependencies. This substitution is useful when only resources are being changed, because it avoids an unnecessary link.

Windows Resource Limits

Number of Windows

There is a limit to the number of windows that can be created systemwide. Each window creates a window structure that is allocated out of USER's heap, a data segment that is limited to 64K. This is the limitation. Since there are a number of items that affect the USER's heap, there is no way to determine how much space will be around at any given time. You can use Heap Walker to check the size of USER's data segment.

How Menus Affect Resources

Menus are stored in GDI's heap and they have a large effect on system resources available. If an application loads a menu and doesn't associate it with a window, it must destroy the menu itself. (Menus used with the **TrackPopupMenu** function won't automatically be freed when the application exits.)

Calculating System Resources

The Free System Resources figure in the Program Manager's "About" dialog box is calculated as follows:

$$\%\ \text{free} = \min(\%\ \text{free of USER's heap},\ \%\ \text{free of GDI's heap})$$

$$\%\ \text{free USER or GDI heap} = \frac{(\text{Max heap size} - \text{heap used})}{\text{Max heap size}} * 100$$

The calculation is the same for all modes (although not done for real mode).

The maximum heap size is 64K less the sum of the size of the static variables and the stack.

There is no external API defined to calculate these values.

IN SUMMARY

Property lists in Windows provide an excellent method for storing a collection of 16-bit data elements that are associated in some way with a window or dialog box. Although a bit slower to get and set than **cbWinExtra** data, property lists are especially valuable when you don't know initially how much data will be required. Property lists can also eliminate the need to construct complex data structures—let Windows do the work for you! PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

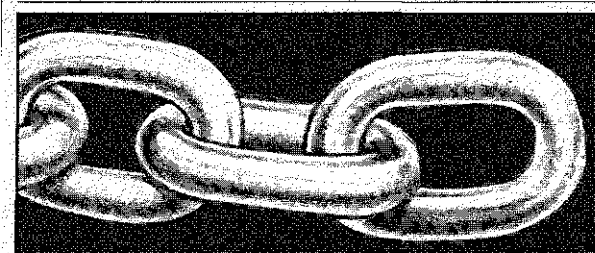
LISTING 1

```
/* Header file for the Windows property list demo
 * Written by Bill Hall */

/* This trick ensures that globals are declared in only one place */
#if !defined(EXTERN)
#define EXTERN extern
#endif

/* Resource string constants */
#define IDS_APPNAME      100
#define IDS_ICON 101
#define IDS_TITLE        102

/* menu items */
#define IDM_ALLOC        100
#define IDM_PROP 101
#define IDM_ABOUT        102

/* dialogs */
#define DT_ABOUT 1

/* Control window id's */
#define IDD_ALLOC        1
#define IDD_ENUM 2

/* Global variables */
EXTERN HWND hWndMain;            /* main window handle */
EXTERN char szAppName[20];       /* name of the application */
EXTERN HWND hWndList;            /* memory allocations display window */
EXTERN HWND hWndEnum;            /* property list display window */
EXTERN HANDLE hInst;             /* instance handle */

/* Declarations which need to be known in several C source files */
LONG FAR PASCAL MainWndProc(HWND,unsigned,WORD,LONG);
BOOL FAR InitProgram(HANDLE, HANDLE, LPSTR, int);
int FAR PASCAL EnumPropCallBack(HWND hWnd, WORD nDummy, PSTR str, HANDLE
                                hData);
BOOL FAR WndCreate(HWND hWnd);
void FAR _cdecl dbs(const char *fmt, ...);

/* memory allocation and property list functions */
void FAR UpdateList(HWND hWnd);
LPSTR FAR AllocMem(HWND hWnd, int size);
void FAR FreeMem(HWND hWnd);

/* Simple data structure to be stored in each chunk of memory allocated */
typedef struct _MYDATA {
    BYTE data[16];
} MYDATA;
```

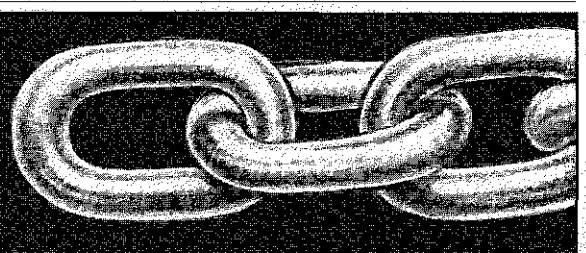
LISTING 2

```
/* Initialization segment for the Windows property list demo
 * This code is discarded after it is used. */
#define NOCOMM
#define NOKANJI
#define NOATOM
#define NOSOUND
#include <windows.h>
#include <string.h>
#include "walloc.h"

/* local function declarations */
static BOOL NEAR RegisterWindowClass(HANDLE);
static void NEAR GetPrevInstanceData(HANDLE);
static HWND NEAR MakeAndShowMainWnd(HANDLE, int);
```

continued



THE MISSING LINKER!



NEW OPTLINK/WINDOWS: THE WORLD'S FASTEST WINDOWS LINKER.

Using a Windows linker isn't tedious and time-consuming anymore. With new OPTLINK/Windows 2.50, it's a snap!

OPTLINK/Windows is the fastest, most powerful linker available for Microsoft Windows. It performs up to 20 times faster than MS-LINK, generating DOS, Windows and OS/2 .EXE files.

Power Without Limits.

Blazing speed is only the beginning. OPTLINK/Windows is the only linker that sets *no* internal limits on the number of modules, libraries, publics or segments you link. OPTLINK/Windows supports CodeView 3.0 & 3.1, and lets you link and debug larger applications than any other linker.

Plus, OPTLINK/Windows utilizes EMS and XMS, and works wonders on a Novell network.

Look At What You're Missing.

OPTLINK/Windows is more than fast and powerful. It's packed with features and functions you can't find anywhere else:

- Copies .RES files to .EXE at link time
- Generates complete global cross-references
- Generates smaller files than MS-LINK
- Detailed segment mapping
- Compatible with all compilers using Intel/Microsoft .OBJ files

Don't Miss Out! Order Now!

OPTLINK/Windows 2.50 is priced at only $350. And it comes with a 30-day money-back guarantee.

Why waste your time with slow, stone-age linkers? Order OPTLINK/Windows now, and get what you've been missing!

SLR

SLR SYSTEMS, INC.

(412) 282-0864 Voice

1622 N. Main St., Butler, PA 16001
(412) 282-7965 FAX ■ (412) 282-2799 BBS

Trademark/Owner: OPTLINK/Windows/SLR Systems, Inc; Windows, MS-LINK, CodeView/Microsoft; Novell/Novell, Inc.; OS/2/IBM.

Circle 34 on Reader Service Card

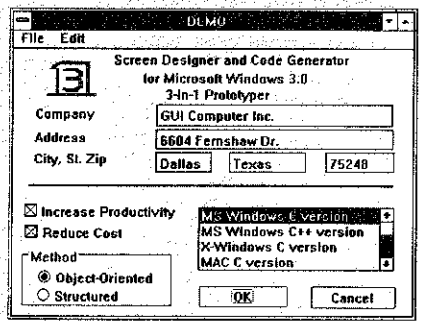
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C/C++ Windows 3.0

Screen Designer Code Generator

Only $159

The easiest way to learn and to master Microsoft Windows programming



With 3-in-1, developing this window screen requires only

20 Minutes !!

That includes screen layout, code generation, compiling and linking.

GUI's 3-in-1 C $159 C++ $199 Both $259

30 Day Guarantee No Source Code Royalties

Also many low cost Windows source code utilities.

1-800-800-9010 TEL 214-250-3472 Fax 214-250-1355

Circle 35 on Reader Service Card

Announcing version 2:

Victor Image Processing Library

Use Victor to develop powerful image applications

Work with Images of any size -- use conventional, expanded, and extended memory

Now your applications can support 8-bit color and gray scale images of any size because Victor gives you complete control over conventional, expanded, and extended memory.

Image-based applications can be developed in MSC, QuickC, and Turbo C environments.

Display on Super VGA

Display images on EGA/VGA and super VGA up to 1024 x 768 256 colors.

Load & save PCX/TIFF/GIF/BIN

Handle images from any source, or create translation programs between the popular file formats.

Your application will be able to print halftones at any size.

Gray scale and color images

Powerful image processing for all images -- your software can have features like: zoom, resize, brighten, contrast, sharpen, outline, linearize, matrix conv, colorize, & more.

ScanJet and LaserJet support

You can offer device control for gray scale scanning -- AND print halftones at any size.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available.

Give your applications support for scanner and laser printer.

***Victor Library version 2*, $195**

Call (314) 962-7833 to order VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 962-7833

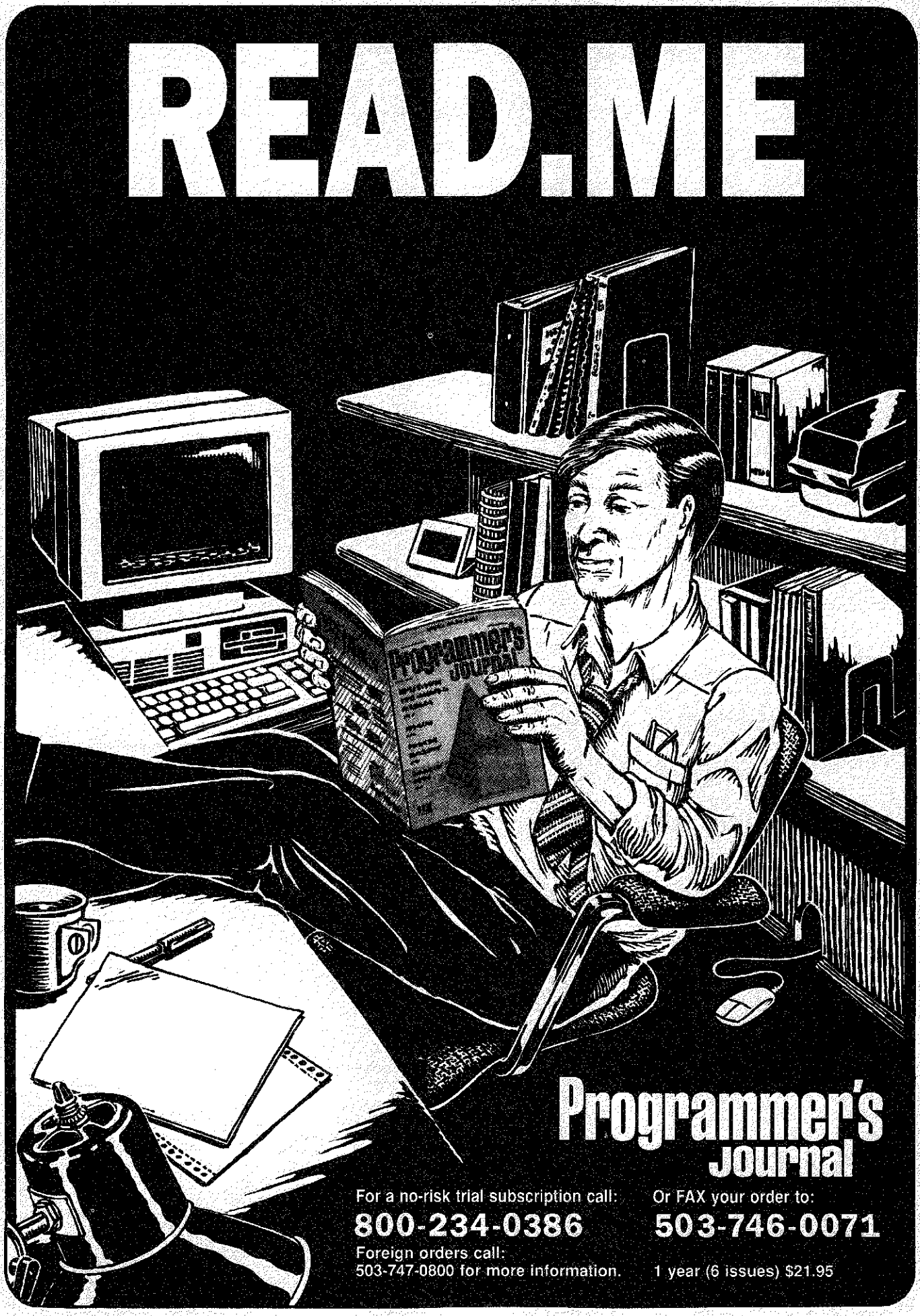
Circle 36 on Reader Service Card


```
/* This routine is FAR since it is called from another segment */
BOOL FAR InitProgram(HANDLE hInstance, HANDLE hPrevInstance,
                     LPSTR lpszCmdLine, int cmdShow)
{
    hInst = hInstance;
  /* if this is the first instance of the program ... */
    if (!hPrevInstance) { /* register the window */
          if (!RegisterWindowClass(hInstance))
            return FALSE;
    }
  /* A previous instance already exists so get global data from there */
    else
         GetPrevInstanceData(hPrevInstance);

  /* Create and show the window */
    if (!(hWndMain = MakeAndShowMainWnd(hInstance, cmdShow)))
         return FALSE;

  /* Show window if child windows created successfully */
    if (hWndList && hWndEnum) {
        ShowWindow(hWndMain, cmdShow);
        UpdateWindow(hWndMain);
    }
    else /* failed, terminate app */
         PostMessage(hWndMain, WM_SYSCOMMAND, SC_CLOSE, 0L);
    return TRUE;
}

/* Every window must belong to a class. We register ours here */
static BOOL NEAR RegisterWindowClass(HANDLE hInstance)
{
    WNDCLASS WndClass;

  /* Load the name string from resources */
    LoadString(hInstance, IDS_APPNAME,(LPSTR)szAppName,sizeof(szAppName));

    memset(&WndClass, 0, sizeof(WndClass));

  /* fill the structure */
    WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);  /* standard cursor */
    WndClass.hIcon = LoadIcon(hInstance, szAppName);
    WndClass.lpszMenuName = (LPSTR)szAppName;        /* get menu */
    WndClass.lpszClassName = (LPSTR)szAppName;       /* our class name */
    WndClass.hbrBackground = COLOR_WINDOW + 1;       /* system color */
    WndClass.hInstance = hInstance;                  /* instance handle */
    WndClass.style = CS_VREDRAW | CS_HREDRAW; /* standard redraw values */
    WndClass.lpfnWndProc = MainWndProc;       /* pointer to window proc */

    return RegisterClass(&WndClass); /* try to register the class */
}

/* If this not the first instance, we can retrieve static data from a
   previous invocation of the program */
static void NEAR GetPrevInstanceData(HANDLE hPrevInstance)
{
    GetInstanceData(hPrevInstance, szAppName, sizeof(szAppName));
}

/* Create the main window */
static HANDLE NEAR MakeAndShowMainWnd(HANDLE hInstance, int cmdShow)
{
    char szTitle[50];

    LoadString(hInstance, IDS_TITLE, szTitle, sizeof(szTitle));

  /* create the window, letting it fall where Windows wants */
    return CreateWindow(szAppName,
                        szTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,0,
                        CW_USEDEFAULT,0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL);
}

BOOL FAR WndCreate(HWND hWnd) /* create the child listboxes */
{
    static char *class = "LISTBOX";

    LONG style = WS_CHILD | WS_VISIBLE | WS_BORDER
                 | WS_VSCROLL | LBS_USETABSTOPS;
    hWndList = CreateWindow(class,
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

READ.ME



Programmer's Journal

For a no-risk trial subscription call:
800-234-0386
Foreign orders call:
503-747-0800 for more information.

Or FAX your order to:
503-746-0071
1 year (6 issues) $21.95

59153

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TIFF & PCX
SDK FOR WINDOWS 3.0

You Can Add TIFF & PCX support for your Windows 3.0 Application without learning TIFF & PCX file formats:

TIFF SDK For Windows is a complete set of easy to use tools to develop TIFF 5.0 support to your application without ever having to learn the complexity of Tag Image File Format. Some of the features are Group 3, Group 4, LZW, Packbit compression, color images, chained images and device independent bitmaps. Comes with a complete set of documentation and a sample program to demonstrate how to use TIFF SDK. **Only $**299.95

PCX SDK For Windows is a complete set of easy to use tools to develop PCX file format for your application without ever having to learn the complexity of PCX file format. It comes with a complete set of documentation and a sample program with the source code to demonstrate how to use PCX SDK. **Only $**199.95

IMAGE SDK For Windows is a complete set of functions to export, import, convert, rotate, invert, and manipulate bitmap images off or on the sreen. It comes with a complete set of documentation and a sample program with the source code to demonstrate how to use IMAGE SDK. **Only $**199.95

CALL for information on NEW Windows Tools.

Software Inc.

TEL: (914) 764-1022 FAX:(914)764-1023

VISA/M.C./C.O.D/CHECK

BLACK ICE SOFTWARE INC.

217 Cedar Hill Lane, Pound Ridge, N.Y. 10576

Circle **38** on Reader Service Card

If your Windows application needs images, you need ImageMan!

ImageMan provides a complete library for displaying and printing virtually any type of image. Our object-oriented approach allows your application to access all images in the same manner -- regardless of the type of image, your code doesn't change!

- Saves months of development effort.
- Use with any Windows 3.0 environment that supports DLLs, including: C, Pascal, Actor, Toolbook, Plus, etc...
- Completely supports TIFF, PCX, EPSF, WMF, and BMP images (with more to come....)
- Easily extended to support new image formats.
- Uses assembly language routines for *Speed.*
- Highly configurable - you have *full* control.
- Royalty Free!

Call today for free demo!

DTI Data Techniques, Inc.
1000 Business Center Drive, #120
Savannah, GA 31405

$395
$995 w/Source

800-868-8003
912-651-8003

Circle **39** on Reader Service Card


```
                            NULL,
                            style,
                            0,0,0,0,
                            hWnd,
                            IDD_ALLOC,
                            hInst,
                            NULL);

    hWndEnum = CreateWindow(class,
                            NULL,
                            style | LBS_SORT,
                            0,0,0,0,
                            hWnd,
                            IDD_ENUM,
                            hInst,
                            NULL);

    return (hWndEnum || hWndList);
}
```

LISTING 3

```
/* Resident segment for the Windows Property List Demo */
#define NOCOMM
#define NOKANJI
#define NOSOUND
#include <windows.h>
#include <stdarg.h>
#include <string.h>
#include <stdio.h>

/* all global variables are declared in this module */
#define EXTERN
#include "walloc.h"

/* local function declaration */
BOOL FAR PASCAL AboutDlg(HWND hDlg, unsigned message, WORD wParam,
                         LONG lParam);

/* Entry point for program */
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdLine, int cmdShow)
{
    MSG msg;

  /* If initialization is not successful then exit */
    if (!InitProgram(hInstance,hPrevInstance, lpszCmdLine, cmdShow))
        return FALSE;

    while (GetMessage((LPMSG)&msg,NULL,0,0)) { /* Retrieve messages */
        TranslateMessage((LPMSG)&msg);
        DispatchMessage((LPMSG)&msg);
    }
    return msg.wParam;          /* exit program */
}

/* All messages are processed here */
long FAR PASCAL MainWndProc(register HWND hWnd, unsigned message,
                            WORD wParam, LONG lParam)
{
    MYDATA FAR *ptr;
    static unsigned count;
    FARPROC fp;

    switch(message) {
        case WM_CREATE:
            WndCreate(hWnd);  /* create the listbox display windows    */
            break;
        case WM_DESTROY:
            FreeMem(hWnd);    /* free allocated memory & property list */
            PostQuitMessage(0);
            break;
        case WM_SIZE:         /* size the list box display windows     */
            if (wParam != SIZEICONIC) {
                MoveWindow(hWndList,0,0,LOWORD
                           (lParam),HIWORD(lParam) / 2,TRUE);
                MoveWindow(hWndEnum, 0, HIWORD(lParam) / 2,
                           LOWORD(lParam), HIWORD(lParam) / 2, TRUE);
            }
            break;
        case WM_COMMAND:
            switch(wParam) {
                case IDM_ABOUT:         /* show about box */
                    fp = MakeProcInstance(AboutDlg, hInst);
                    DialogBox(hInst,MAKEINTRESOURCE(DT_ABOUT),hWnd,fp);
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                FreeProcInstance(fp);
                break;
            /* allocate a chunk of data; if successful,
               copy in some information and display the result */
                case IDM_ALLOC:
                    ptr = (MYDATA FAR *)AllocMem(hWnd, sizeof(MYDATA));
                    if (ptr) {
                        wsprintf(ptr->data, "%s%u", (LPSTR)"Mydata",
                            count++);
                        UpdateList(hWnd);
                    }
                    break;
            /* enumerate the current property list and display
               it in a listbox.  To keep the display from
               flashing during update, turn off the redraw flag
               before the update and restore it when done. */
                case IDM_PROP:
                    SendDlgItemMessage(hWnd,IDD_ENUM,
                                       WM_SETREDRAW,FALSE,0L);
                    SendDlgItemMessage(hWnd,IDD_ENUM,
                                       LB_RESETCONTENT,0,0L);
                    fp = MakeProcInstance(EnumPropCallBack, hInst);
                    EnumProps(hWnd, fp);
                    FreeProcInstance(fp);
                    SendDlgItemMessage(hWnd,IDD_ENUM,
                                       WM_SETREDRAW,TRUE,0L);
                    InvalidateRect(hWndEnum, NULL, TRUE);
                    UpdateWindow(hWndEnum);
                    break;
            }
            break;
        default:
            return ((long)DefWindowProc(hWnd,message,wParam,lParam));
            break;
    }
    return(0L);
}

/* This function allows you to print debugging information
 * to a secondary monitor attached to a monochrome display
 * card, Hercules board, or equivalent.
 *
 * To use this function you must load the device driver OX.SYS
 * Do NOT run Windows with redirection to AUX.
 *
 * Use dbs just like printf.
 * Example dbs("function foo %d %s\r\n", myint, mystring); */
void FAR _cdecl dbs(const char *fmt, ...)
{
    char buf[255];

    va_list arg_ptr;

    va_start(arg_ptr, fmt);
    vsprintf(buf, fmt, arg_ptr);
    va_end(arg_ptr);

    OutputDebugString(buf);
}

/* show ego box */
BOOL FAR PASCAL AboutDlg(HWND hDlg,unsigned message,WORD wParam,
                         LONG lParam)
{
    switch(message) {
        case WM_INITDIALOG:
            break;
        case WM_COMMAND:
            switch(wParam) {
                case IDOK:
                case IDCANCEL:
                    EndDialog(hDlg, TRUE);
                    break;
                default:
                    return FALSE;
            }
            break;
        default:
            return FALSE;
    }
    return TRUE;
}
```

continued

We Have Ways To Improve Your Productivity

TE Developer's Kit

Why reinvent the wheel? TE Developer's Kit provides the most cost effective way of incorporating text editing features into your application. The kit includes the complete source code for the editor and the executable file. Supports Microsoft and Borland 'C' compilers.

Features: multiple files/windows, word-wrap and printing, edits files larger than available memory, reconfigurable keyboard commands and screen colors, undo, column block commands, and search/replace. **The kit also includes TER small editor routine. TER is a subset of TE. Your application calls this routine by specifying the window location and size, maximum file size, and an input buffer or file.** *(DOS: $169, Windows: $269)*

WinMem: Windows 3 Memory Manager

WinMem provides an unlimited amount of virtual memory even in Windows' Real and Standard modes. **The device independence is not enough. WinMem offers you CPU and memory resource independence.** It provides an unlimited number of global memory handles, and reduces the overhead per block to 4 bytes (from 24 bytes of Windows). *($139)*

Spell Time *(NEW)*

Spell Time consists of a dictionary (over 100K words) and the routines to access the dictionary. Also includes an application specific dictionary, and a user dictionary. The routine will suggest alternates for a misspelled word. Highly optimized: 50 to 75 words/sec on a 12 Mhz 286 computer. Uses virtual memory on disk or EMS. An interface with TE included. Includes the complete source code. *($369)*

Sub Systems, Inc. 1-800-447-6819
Fax: 1-617-438-0311
159 Main Street, #8C, Stoneham, MA 02180, 1-(617)-438-8901

Circle **40** on Reader Service Card

The Fast, Flexible Easy to Use GUI

TEGL WINDOWS TOOLKIT II

Programmers, now you can **create true event-driven stand-alone GUI** (graphical user interface) applications for DOS. Applications will not need any windows operating environment. Outstanding features include: local menus; resizeable windows; sliders; scrolling; mouse/keyboard/timer interrupt handlers; virtual memory manager that uses EMS/hard disk and handles; dialogue management; 49 fonts; world coordinates; Turbo Pascal objects and Turbo C++ class library.

Font and icon editors provide the tools to customize the finest details of your programs. TEGL Graphic Interface provides BGI compatibility with greater speed. **No runtime modules nor royalty payments** are required on applications developed with TEGL. The Toolkit adds as little as 100K to the size of an application. The **complete source code** is provided.

Supported graphics modes (colors): CGA(2), Hercules(2), EGA(16), VGA(16) and Super VGA(16, 256).

Supported compilers: Turbo Pascal, Stony Brook Pascal+, Turbo C®, Turbo C++®, Borland C®, Quick C®, Microsoft®C, WATCOM® C, and JPI Topspeed C.

****

TEGL WINDOWS TOOLKIT II IS ONLY $99. *ORDER NOW!*
Phone: (604) 669-2577 **Fax: (604)688-9530**

****

Shipping & Handling $10 ($15 outside North America)
Visa, Mastercard, Cheque

TEGL Systems Corporation
780-789 West Pender Street
Vancouver, BC, Canada V6C 1H2

30 day money back guarantee

Trademarks are property of their respective holders.

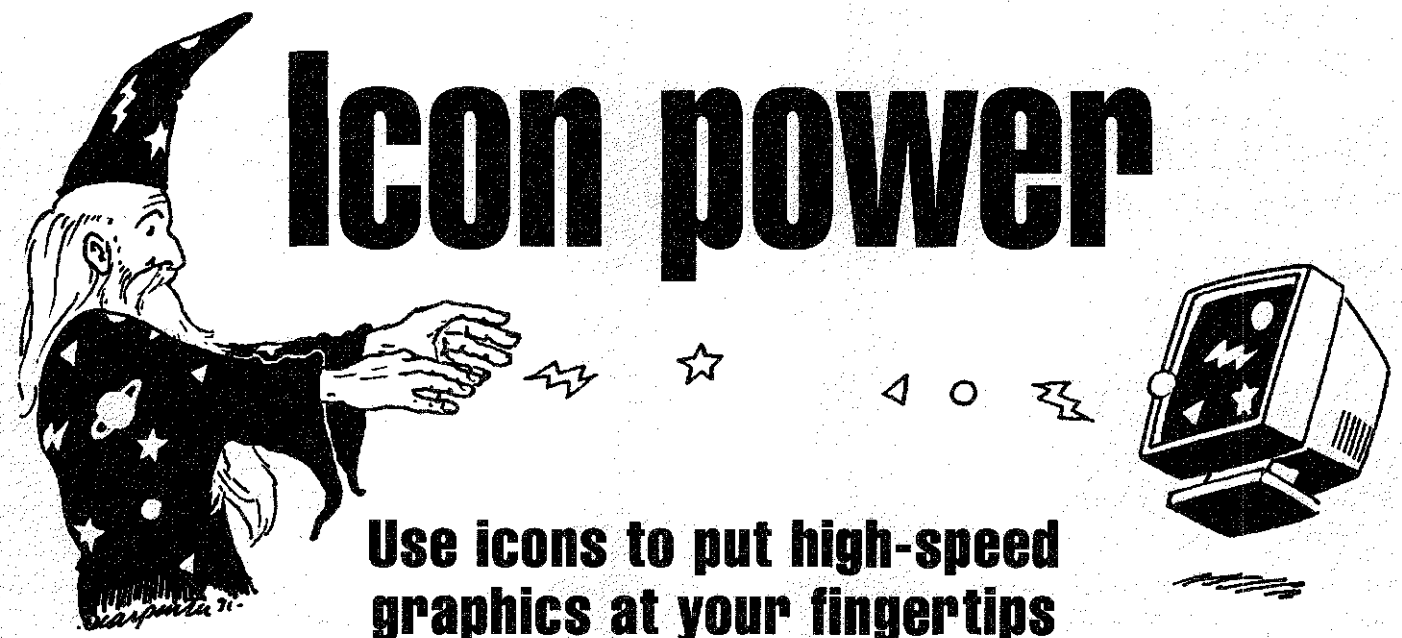
Circle **41** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 4

```
/* Memory allocation and property list functions for
 * the Windows property list demo */
#define NOCOMM
#define NOKANJI
#define NOSOUND
#include <windows.h>
#include "walloc.h"

/* local function declaration */
static LPSTR NEAR WirePointer(DWORD size);

/* property list strings */
static char *szCurBlockNum = "CURBLOCKNUM";
static char *szCurSlotNum = "CURSLOTNUM";
static char *szCurSelector = "CURSELECTOR";

/* locally shared variables */
static int slotsize;

/* definitions */
#define SLOTS_PER_BLOCK  5

/* Allocate a memory chunk and store the information
 * on the window property list.  Memory is allocated in
 * a block whose size depends on the number of slots
 * desired and the size of an individual chunk.  It
 * is assumed that a chunk of memory always has the
 * same fixed size. */
LPSTR FAR AllocMem(HWND hWnd, int size)
{
    WORD CurBlockNum;    /* current block and slot numbers */
    WORD CurSlotNum;
    LPSTR memptr;        /* memory pointer */
    WORD sel;            /* selector (or segment) value */
    WORD offset;         /* pointer offset */
    char buf[30];        /* space for property strings */
    WORD handle;

    CurBlockNum = GetProp(hWnd, szCurBlockNum);   /* get current block */
    CurSlotNum = GetProp(hWnd, szCurSlotNum);     /* and slot number */

    /* we are either starting off the allocation or we have to
       create a new block */
    if ((CurBlockNum == 0) ||((CurBlockNum > 0) &&
        (++CurSlotNum == SLOTS_PER_BLOCK))) {
        slotsize = size;
        /* keep the pointer for the life of the program */
        memptr = WirePointer(size * SLOTS_PER_BLOCK);
        if (memptr) {
            sel = HIWORD(memptr);        /* break pointer into components */
            offset = LOWORD(memptr);
            CurBlockNum += 1;            /* increment block number */
            CurSlotNum = 0;              /* new slot number is 0 */
        /* remove old props before changing values
         * this step is not necessary but improves
         * the display during EnumProps */
            RemoveProp(hWnd, szCurBlockNum);
            RemoveProp(hWnd, szCurSlotNum);
            SetProp(hWnd, szCurBlockNum, CurBlockNum); /* set the new ones */
            SetProp(hWnd, szCurSlotNum, CurSlotNum);
            /* make a unique string identifying the block */
            wsprintf(buf, "%s%x", (LPSTR)szCurSelector, CurBlockNum);
            if (!SetProp(hWnd, buf, sel)) {      /* save to property list */
                handle = LOWORD(GlobalHandle(sel));/* failed, free allocation */
                GlobalUnWire(handle);
                GlobalFree(handle);
                memptr = NULL;
            }
        }
    }
    /* we are simply adding a new slot to the current block */
    else {
        /* build string identifying the current block */
        wsprintf(buf, "%s%x", (LPSTR)szCurSelector, CurBlockNum);
        sel = GetProp(hWnd, buf);        /* get the selector */
        offset = CurSlotNum * size;      /* create the offset */
        memptr = (LPSTR)MAKELONG(offset, sel);  /* make the pointer */

    /* this next step is not really necessary but
     * it improves the display during EnumProps */
        RemoveProp(hWnd, szCurSlotNum); /* update current information */
        SetProp(hWnd, szCurSlotNum, CurSlotNum);
    }
    return memptr;
}

/* free all allocated memory and remove the property list items */
void FAR FreeMem(HWND hWnd)
{
    register WORD i;
    WORD CurBlockNum = GetProp(hWnd, szCurBlockNum);
    WORD cursel;
    char buf[30];
    WORD handle;
    BOOL result;

    for (i = 1; i <= CurBlockNum; i++) {
        /* create the property list identifier */
        wsprintf(buf, "%s%x", (LPSTR)szCurSelector, i);
        cursel = GetProp(hWnd, buf);     /* get the selector */
        handle = LOWORD(GlobalHandle(cursel));  /* convert to handle */
        result = GlobalUnWire(handle);  /* release the allocation */
        result = GlobalFree(handle);
        RemoveProp(hWnd, buf);          /* remove the property */
    }
    RemoveProp(hWnd, szCurBlockNum);     /* remove remaining properties */
    RemoveProp(hWnd, szCurSlotNum);
}

/* add the most recent allocation in the allocation list box */
void FAR UpdateList(HWND hWnd)
{
    WORD CurBlockNum, CurSlotNum;
    char buf[80];
    int pos;
    WORD sel, offset;
    MYDATA FAR *ptr;

    /* get the current block and slot numbers */
    CurBlockNum = GetProp(hWnd, szCurBlockNum);
    CurSlotNum = GetProp(hWnd, szCurSlotNum);

    /* build a pointer to the data in a slot */
    wsprintf(buf, "%s%x", (LPSTR)szCurSelector, CurBlockNum);
    sel = GetProp(hWnd, buf);
    offset = CurSlotNum * slotsize;
    ptr = (MYDATA FAR *)MAKELONG(offset, sel);

    /* build a string to be inserted into the listbox */
    if (CurSlotNum == 0)
        wsprintf(buf, "Block: %2u\tSlot: %2u\tPointer: %x:%x\tData: %s",
                 CurBlockNum, CurSlotNum, sel, offset, (LPSTR)ptr->data);
    else
        wsprintf(buf, "\tSlot: %2u\tPointer: %x:%x\tData: %s",
                 CurSlotNum, sel, offset, (LPSTR)ptr->data);

    /* insert into listbox */
    pos = (int)SendDlgItemMessage(hWnd, IDD_ALLOC,
                                  LB_ADDSTRING, 0, (LONG)(LPSTR)buf);
    /* highlight the new entry */
    if (pos != LB_ERR)
        SendDlgItemMessage(hWnd, IDD_ALLOC, LB_SETCURSEL, pos, 0);
    else
        MessageBeep(0);
}

/* allocate global data and wire the pointer */
static LPSTR NEAR WirePointer(DWORD size)
{
    HANDLE hmem = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT, size);
    if (hmem)
        return GlobalWire(hmem);
    else
        return (LPSTR)NULL;
}

/* enumerate the property list and display in listbox */
int FAR PASCAL EnumPropCallBack(HWND hWnd, WORD nDummy,
                                PSTR str, HANDLE hData)
{
    char buf[80];

    if (nDummy != 0)
        wsprintf(buf, "%x\t%s\t%x", hWnd, (LPSTR)str, hData);
    else
        wsprintf(buf, "%x\t%x\t%x", hWnd, str, hData);
    SendDlgItemMessage(hWnd,IDD_ENUM,LB_ADDSTRING,0,(LONG)(LPSTR)buf);
    return TRUE;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Icon power

Use icons to put high-speed graphics at your fingertips

By Peder Jungck

In my last article ("Pixel panning," *PJ* 9.3), I explained how to use pixel panning and how to remap the video adapter memory to create useful effects. This article examines another tool—icons—and discusses how icons can be useful in the development of professional graphics software.

When icons are mentioned, most people think of Microsoft Windows or the Macintosh: graphical user interfaces have defined icons as things that represent specific objects. But an icon can also be defined as a collection of pixels that forms an image. As such, they are excellent building blocks for full-screen graphical applications. Rather than drawing many different full-screen images in real time, you can generate several small images—namely, icons—that can be pieced together to create a full-screen display. The use of icons can speed the development and runtime of graphics generation, reduce memory use, and simplify complex images. If you're not familiar with the benefits of icon use, take a look at some video games and note the difference between background and foreground objects. Generally, the foreground consists of moving icons, and the background is a repeated pattern of fixed icons.

Like many programmers, I am not a graphic artist, and the use of icons as building blocks has improved the visual quality of my applications. Using icons, you can focus on drawing a single image. If it doesn't look right, you can modify it until you like what you see. A composite of these icons presents an impressive full-screen image that is easy to generate.

Drawing icons can be especially speedy if you predetermine fixed data sizes. When fixed data sizes are used, the display code doesn't have to adapt dynamically. If one icon can be repeated to form a pattern, you can save RAM and, with a few tricks, improve display speed.

There are times, however, when drawing a screenful of data from system RAM to the video board using icons is slower than using a single, large image. For example, the entry and exit code used by the icon display routines will slow image generation if you are copying many different icons to fill the display. Because a single image display routine does not have the entry and exit code overhead for the same amount of data, it can be faster. So don't throw away those other icon libraries—a full-screen image paint routine can sometimes be the best option.

USE THE WRITE MODE

Depending on their design and implementation, your icons can often be streamlined. If you have small objects or much repetition of patterns, you can place one of each icon into off-screen video memory and then use the EGA/VGA write mode 1 to do a very fast move. When you read from a memory location in write mode 1, all four planes of data are moved into their corresponding latches. Each plane is 64K, and all the planes have the same segment address. A subsequent write takes the data from the latches and copies it to a new memory location. Because the data being moved stays in the latches on the video board, the main bottleneck to placing data in video RAM—the bus—is avoided. Using a single **movsb** statement, four bytes of data can be copied into the latches and then to the new memory location (see **Copy_Icon,** Listing 1).

Another benefit of using this feature of write mode 1 is that the video board does not care what the settings are in the Set/Reset, Data Rotator, ALU, and Bit Mask registers; thus, they need not be set before the data is copied. The Map Mask register is still active, so you can select specific planes. These registers do not affect the data, because the data must flow from the system RAM

Peder Jungck, a software engineer, is the author of the ProGraphx Toolbox EGA/VGA graphics library. He can be reached at 1019 West Lovers Lane, Arlington, TX 76013.

ILLUSTRATION BY JIM CARPENTER

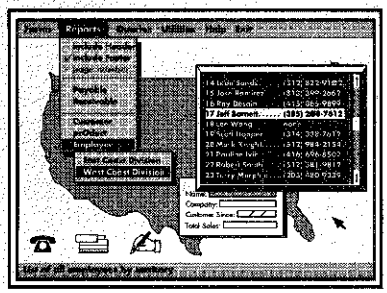
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IF YOU PROGRAM
in C, C++, Basic, Fortran, Cobol, Pascal, dBase, *.*

HI-SCREEN Pro II™
is your user interface solution

Our new HI-SCREEN Pro II is an integrated environment for creating and managing user interfaces for your programs.

- Built-in editors allow you to interactively **generate objects** (windows, menus, icons, mouse cursors, graphs, help screens, etc.)
- Your programs can easily **manipulate** these **objects,** using a set of powerful functions.
- You can **test** all **objects** directly under the editor in any graphic mode (Hercules, CGA, EGA, VGA).
- New features include: New Icon Editor • New Menu Generator • New Graph Editor • New Library Generator • Enhanced scrolling (automatic scroll boxes) • 25/30/43/50 line mode support • Enhanced data validation • etc.



- Supports all versions of **C, Basic, Cobol, Fortran, Pascal, dBase, Assembly,** etc.
- Call for OS/2 and Windows versions.
- Includes linkable modules for C, Pascal, Fortran, Cobol, Clipper and Quick Basic.

HI-SCREEN Pro II **$395**
Upgrades ..... **Call**

30 day money-back guarantee • No royalties

1-800-338-2852
Tel.: 415-896-0708 • Fax: 415-896-0709

Softway, Inc.
185 Berry Street, Suite 5411,
San Francisco, CA 94107

NEW VERSION!

Circle 42 on Reader Service Card

for the hardware to perform these functions. In write mode 1, the read places the data in the latches and then sends it back to the video RAM. This means that the data bypasses the hardware. Because the EGA/VGA does not care about these registers, you can omit all references to them, decreasing code size and improving speed.

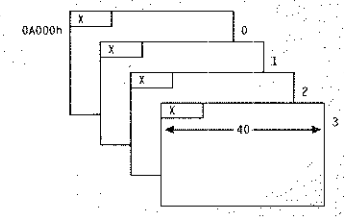
The move routines do have some drawbacks. Because you are only copying data in write mode 1, the data you want to display must already exist somewhere in the video RAM. Additionally, the disregard for register settings precludes masking portions of bytes or shifting the data by pixel increments.

To use the write mode 1 for icon display, you must store the data somewhere in video RAM. The **Display_Icon** routine (see Listing 1) copies an icon from system RAM and then stores it in video RAM. **Display_Icon** takes as parameters the addresses of the icon data, horizontal byte, and vertical scan line. **BytesPerLine** stores the number of bytes per line, because video RAM can be remapped (as discussed in the previous article). The test program uses video mode 13, 320 × 200 × 16 colors, which defaults **BytesPerLine** to 40.

WHICH FORMAT FOR YOU?

Selecting the icon data format that best suits your video mode is as important as choosing the routines that display icons on the screen. To determine the most efficient layout, you must know the layout of the video RAM for the mode you are using. Figure 1 shows the layout for

FIGURE 1:Video RAM layout for EGA mode 13



Mode 13(0dh) (40 bytes per line). One bit per plane per pixel. 320 x 200 with 16 colors.

EGA mode 13; the only difference among video modes 13(0dh), 14(0eh), 16(10h), and 18(12h) is the **BytesPerLine** value. It is most practical to create icons in byte increments, because a byte copied to the video RAM represents eight pixels when the VGA controller is set up for using planes. With one byte per plane, the icon requires four bytes to represent eight pixels.

Whether you should draw the icon one plane at a time or draw all four planes one byte at a time depends on the size of your icon. The primary factor is how long it takes to toggle the Map Mask register to select different planes compared with the time it takes to add the offset to the next line. If your icon is only a few bytes wide, it is usually faster to generate it a byte at a time rather than a plane at a time. Figure 2 shows the icon formats for a 16-pixel × 1-line icon in both the byte and the plane layout methods (as well as in bytes with masking, discussed below).

FIGURE 2: Possible formats for 16-pixel x 1-pixel icon

A byte at a time

Byte 1				Byte 2			
Plane 0	Plane 1	Plane 2	Plane 3	Plane 0	Plane 1	Plane 2	Plane 3

A byte at a time with masking

Byte 1					Byte 2				
Mask	Plane 0	Plane 1	Plane 2	Plane 3	Mask	Plane 0	Plane 1	Plane 2	Plane 3

A plane at a time

Byte 1	Byte 2	Byte 1	Byte 2	Byte 1	Byte 2	Byte 1	Byte 2
Plane 0	Plane 0	Plane 1	Plane 1	Plane 2	Plane 2	Plane 3	Plane 3

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WEITEK

THERE'S ONE REASON FOR HAVING A 486 BASED PC, SPEED

AND THERE'S ONE REASON FOR ADDING A WEITEK 4167, MORE SPEED

If you're working with a 486 based PC, then speed is almost certainly one of the reasons why. And if it is, you're going to want the added speed offered by a Weitek 4167 coprocessor.

Weitek makes the fastest coprocessors in the world for 386 based PCs, and the only coprocessor for 486 machines. These coprocessors dramatically increase the speed of a growing wave of high-end applications like 3-D graphics, CAD/CAM, solids modeling, photorealistic imaging, rendering, multimedia and simulation.

Speed gains of up to 2X and beyond, can be realized with Weitek coprocessors. Not to mention hours of wait time that can be eliminated as well.

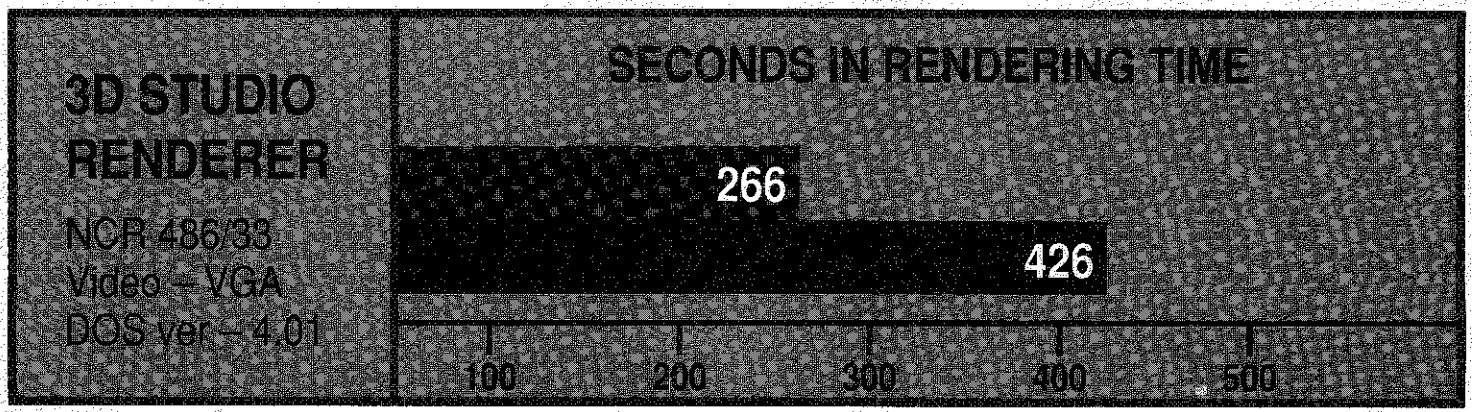
CAD software from Autodesk, Versacad and CADKEY, plus compilers from Metaware, Microway and SVS, all run best with Weitek coprocessors.

What makes all this significant is that taking advantage of Weitek's coprocessors is so simple. The chip is extremely cost effective, and it installs easily. Virtually every manufacturer of 386 and 486 machines supports the 3167 or 4167. AMI, AST, Compaq, DEC, NEC, Olivetti, and a great many more.

To start taking advantake of Weitek's 2X performance improvement, contact your PC dealer or reseller. Or call us at 1-800-HOT-3167.

WEITEK 4167 vs Intel Performance Benchmarks

WEITEK Intel



Circle 43 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
/* ICONDEMO.c --- a sample program for drawing and moving icons          */

#include <stdio.h>
#include <stdlib.h>
typedef char Icon[160];           /* array of 160 bytes forms an icon.  */
/* display icon at x,y from system RAM into video RAM with write mode 0 */
extern void near Display_Icon(int x, int y, void far *Data);
/* copy icon from one spot in VRAM to another spotwith write mode 1.    */
extern void near Copy_Icon(int OrigX, int OrigY, int X, int Y);
extern void near Video_Mode(int Mode) ;
extern void near Set_Pixel(int x, int y, int Color);

Icon Icon1 = {
              0x03,0x00,0x00,0x00,0x00, 0xF8,0x00,0x00,0x00,0x00,
              0x07,0x00,0x03,0x03,0x03, 0xFC,0x00,0xF8,0xF8,0xF8,
              0x0F,0x00,0x07,0x07,0x07, 0xF8,0x70,0x70,0x70,0x80,
              0x0F,0x00,0x06,0x06,0x07, 0xF8,0xF0,0xD0,0xD0,0x40,
              0x0F,0x00,0x06,0x06,0x07, 0xFC,0xF8,0xF8,0xF8,0x00,
              0x0F,0x01,0x05,0x05,0x06, 0xF8,0xE0,0xE0,0xE0,0x10,
              0x0F,0x01,0x05,0x05,0x06, 0xF8,0x70,0x70,0x50,0xB0,
              0x0F,0x01,0x01,0x01,0x00, 0xFC,0xB8,0xB8,0xB8,0x40,
              0x3F,0x00,0x00,0x0F,0x02, 0xFE,0x00,0x00,0xCC,0x0C,
              0x7F,0x00,0x00,0x3F,0x21, 0xFE,0x00,0x00,0xFC,0x0C,
              0x7F,0x00,0x00,0x33,0x30, 0xFC,0x00,0x00,0xF0,0xCC,
              0x3F,0x00,0x00,0x03,0x00, 0xFC,0x00,0x00,0xC0,0x00,
              0x3F,0x18,0x18,0x1B,0x1B, 0xFE,0x04,0x04,0xF4,0x04,
              0x7F,0x30,0x30,0x3F,0x30, 0xFE,0x0C,0x0C,0xFC,0xCC,
              0x7F,0x20,0x20,0x27,0x20, 0xFC,0x18,0x18,0x18,0x18,
              0x27,0x00,0x00,0x00,0x00, 0x18,0x00,0x00,0x00,0x00
            };
void delay(int count);

void main ()
{
    int i,j;                          /* loop variables               */

    Video_Mode ( 0x0d );       /* Step 1-set video mode to 320x200x16   */
                               /* Sets up 2 wide by 4 high memory       */
    for (j=0; j<=15; j++)      /* Step 2-manually draw the background   */
        for (i=0; i<=15; i++)  /*   icon using the Set_Pixel function   */
            if ((i+j & 1)==1)  /* Alternate pixels lt. and dk. blue     */
                Set_Pixel(i,j,1);  /* This generates the background     */
            else Set_Pixel(i,j,9); /* pattern, which you can change.    */

    for (j=0; j<13; j++)       /* Step 3-copy the icon to fill screen   */
        for (i=0; i<=19; i++)  /*   12.5 vertical is actual needed,     */
            Copy_Icon(0,0,i*2,j*16); /* the extra byte is offscreen     */

    Display_Icon(2, 90,&Icon1[0]); /* Step 4-draw icon bitmap to start  */
    Display_Icon(0,184,&Icon1[0]); /* position, draw copy for Copy_Icon */

    getch();                  /* wait for user input                    */
#define DELAY    100          /* set to your liking to slow things down */
    for (j=0; j<20*DELAY; j += DELAY) {   /* add our little delay loop  */
        Copy_Icon(2,90,37,90);    /* redraw the background area we use  */
        for (i=2; i<38; i++) {    /* Step 5-do some animation with icon */
            Copy_Icon(0,184,i,90);      /* copy icon to the new position */
            Copy_Icon(0,  0,i-2,90);    /* overwrite trailing w/bkg      */
            delay(j);                   /*    and gradually slow it down */
        }
        Display_Icon(0,184,&Icon1[0]); /* draw a new copy for Copy_Icon */
   }
    getch();                  /* wait for user prompt                   */
    Video_Mode ( 3 );         /* restore to text mode before exiting    */
};

void delay(int count) {       /* our delay function to slow things down */
    while (count--)
        srand(0);             /* use any function for delay             */
}
```

LISTING 2

```
; ICONTOOL.ASM -- EGA/VGA Icon Drawing
; set up for C style calling method for routines
; these routines are currently set up to use a near call model
; by Peder Jungck, Copyright 1991
;
DATA  segment byte public 'DATA'
DATA  ends

_TEXT  segment byte public 'CODE'
       assume     cs:_TEXT, ds:DATA, es:nothing;

Public _Copy_Icon              ;(int OrigX, int OrigY, int X, int Y);
Public _Display_Icon           ;(int x,int y,void far *Data);
Public _Video_Mode             ;(int Mode);
Public _Set_Pixel              ;(int x,int y,int Color);

BytesPerLine  dw  40           ; screen width in bytes (320x200 Mode)
VOffset       equ 0a000h       ; video RAM offset

Display_IconParms struc
                dw     2 dup(?) ; pushed BP and return address
    pp_xpos     dw     ?
    pp_ypos     dw     ?
    pp_pointer  dw     ?
Display_IconParms ends
_Display_Icon proc near    ; _Display_Icon(int x, int y, void far *Data);
        push    bp
        mov     bp,sp
        push    ds
        push    di
        push    si

        mov     ax,VOffset              ; EGA address
        mov     es,ax                   ; segment ega

        mov     ax,BytesPerLine         ; RAM line length currently set
        xchg    di,ax                   ; BytesPerLine to DI
        lds     si,[bp+pp_pointer]      ; ds:[si]=start of icon

        mov     ax,word ptr [bp+pp_ypos] ; get vertical line
        mul     di

        mov     bx,word ptr [bp+pp_xpos] ; horizontal byte
        add     bx,ax                   ; (y*40) + bx <- horiz byte
                                        ; normally in wm0

        dec     di                      ; next line offset variable
                                        ; adding at position 2 in icon
                                        ; to end up at position 1 in
                                        ; next line of icon in video RAM
        mov     dx,03c4h                ; select the map register
        mov     al,2
        out     dx,al

        mov     cx,16                   ; number of lines per icon
DIStrt: mov     dx,3ceh
        lodsb                           ; get the mask
        mov     ah,8
        xchg    ah,al
        out     dx,ax
        mov     dx,03c5h                ; load map register data addr
        mov     al,1                    ; plane 1
        out     dx,al
        lodsb
        xchg    es:[bx],al

        mov     al,2                    ; plane 2
        out     dx,al
        lodsb
        xchg    es:[bx],al

        mov     al,4                    ; plane 4
        out     dx,al
        lodsb
        xchg    es:[bx],al

        mov     al,8                    ; plane 8
        out     dx,al
        lodsb
        xchg    es:[bx],al

        inc     bx                      ; column 2
        mov     dx,3ceh
        lodsb.                          ; get the mask
        mov     ah,8
        xchg    ah,al
        out     dx,ax

        mov     dx,03c5h                ; load map register data addr

        mov     al,1                    ; plane 1
        out     dx,al
        lodsb
        xchg    es:[bx],al
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INTENSIVE. IN-DEPTH.

PACKED WITH C SOURCE CODE.

C GAZETTE
COMPLETE JOYSTICK INTERFACE
FILE VALIDATION WITH CRC
A CALLABLE GREP ENGINE
CONVERTING PROPORTIONAL LASERJET FONTS TO FIXED WIDTH
PROGRAMMING FOR WINDOWS AND PM
EXPERT SYSTEM ENGINE IN C++
MINIMAL SWAPPING SORT
REVIEW OF TURBO C++
ROUNDUP OF ISAM AND DATABASE LIBRARIES
ANNUAL BOOK REVIEW
THE CODE-INTENSIVE C AND C++ QUARTERLY FOR MS-DOS AND OS/2 SYSTEMS
AUTUMN 1990
$6.50
Algorithms

(& RIGHT NOW, FREE.)

DATA COMPRESSION • HASH TABLES • DOS INTERFACE • MICE • LASER PRINTERS • DEBUGGING • DATA EDITING ALGORITHMS • FONTS • GRAPHICS • TREES • BINARY SEARCH • TELECOMMUNICATIONS • AND MORE!

C Gazette is the only code-intensive C and C++ publication. Each issue offers cover-to-cover code with supporting text. Topics are treated in depth and presented in a clear, cogent manner that emphasizes real-world solutions.

For a **FREE preview copy,** fill in your name and address on the attached card. Or for faster service, call **1-800-234-0386.** We'll send you the next issue of ***C Gazette*** and invoice you for a one-year subscription. If you are not satisfied, write cancel on the invoice. The free issue is yours to keep.

1 year (6 issues) **$24.95**

49109

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ProGraphx Toolbox™

New Version 2.0 Write for a Free Demo Disk

NEW FEATURES INCLUDE:

- 320x200 EGA mode for games
- Icon Animation Editor
- Great for games, real-time,
- or detail-oriented graphics

Over 50 highspeed graphics routines for EGA/VGA 16 color modes, including proprietary color icon routines. **ProGraphx Image Editor** is a complete color icon paint package which allows you to edit and view up to 50 icons at once. Simply paint an icon, then use the **ProGraphx Toolbox™** routines to display it with amazing speed. Over 500 sample icons for use in your own programs. Includes **Operation: Airlift**, a full-screen scrolling arcade style game with source code. Works with ANY C or Turbo Pascal.

All routines written in assembly forfastest performance

SPECIAL OFFER

$99 with source

No Royalties!

To order, write:

Peder Jungck
1019 West Lovers Lane
Arlington, TX 76013
Include $5 S&H

Circle 46 on Reader Service Card

Passive Backplane CPUs

PC Tech manufactures a full line of modular passive backplane high performance PC compatible CPUs.

Passive backplane based systems have many advantages, including ease of assembly, flexibility, fast service (by swapping boards), and simple upgrades to new or faster processors.

Contact us for more information, pricing, and benchmark results.

80386SX: 20 MHz, 512K through 16M DRAM
80386DX: 25 MHz & 33 MHz, 1M - 32M DRAM, 128K cache module optional
80486: 25 MHz & 33 MHz, 1M - 32M DRAM, 256K cache module optional

Call, write, or FAX for complete specifications and retail or OEM terms. All PC Tech products are available in complete systems as well as components for retail, OEM, and private label sales.

Designed, Manufactured, Sold and Serviced by:

PC Tech Inc.

Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

Circle 47 on Reader Service Card


```
        mov     al,2                    ; plane 2
        out     dx,al
        lodsb
        xchg    es:[bx],al

        mov     al,4                    ; plane 4
        out     dx,al
        lodsb
        xchg    es:[bx],al

        mov     al,8                    ; plane 8
        out     dx,al
        lodsb
        xchg    es:[bx],al

        add     bx,di                   ; start of next line of icon
                                        ; in video RAM
        loop    DIStrt                  ; do next row

        mov     al,0fh                  ; reset the map mask
        out     dx,al
        mov     dx,3ceh
        mov     ax,0ff08h               ; reset bit mask to all ones
        out     dx,ax

        pop     si
        pop     di
        pop     ds
        pop     bp
        ret
_Display_Icon endp
Copy_IconParms struc
                dw      2 dup(?) ; pushed BP and return address
    CIOx        dw      ?
    CIOx        dw      ?
    CIx         dw      ?
    CIy         dw      ?
Copy_IconParms ends

_Copy_Icon proc near ; Copy_Icon(int OrigX, int OrigY, int X, int Y);
        push    bp
        mov     bp,sp
        push    si
        push    di
        push    ds

        mov     ax,VOffset              ; EGA address
        mov     es,ax                   ; segment ega
        mov     ds,ax                   ; segment for ega

        mov     ax,word ptr [bp+CIOy]   ; get origination icon addr
        mul     BytesPerLine            ; get vertical line

        mov     si,word ptr [bp+CIOx]   ; horizontal byte
        add     si,ax                   ; (y*40) + bx <- horiz byte

        mov     ax,word ptr [bp+CIy]    ; get vertical line
        mul     BytesPerLine

        mov     di,word ptr [bp+CIx]    ; horizontal byte
        add     di,ax                   ; (y*40) + bx <- horiz byte
                                        ; normally in wm0

        mov     dx,03c4h
        mov     al,02
        out     dx,al

        mov     dx,03c5h                ; load map register data addr

        mov     al,0fh                  ; enable all planes
        out     dx,al

        mov     dx,3ceh
        mov     al,5
        out     dx,al
        inc     dx
        mov     al,1
        out     dx,al                   ; select write mode 1

        mov     ax,BytesPerLine         ; offset to next line
        dec     ax                      ; subtract icon width
        dec     ax

        mov     cx,16
CIStrt: movsb   ; reads ds:[si] then writes es:[di]
        movsb   ; reads icon place in memory then writes to dest

        add     si,ax                   ; get to start of next line of
        add     di,ax                   ; the orig and dest addresses
        loop    CIStrt

        mov     dx,3cfh                 ; write mode register already
        mov     al,0                    ; selected, just output value
        out     dx,al                   ; restore write mode 0

        pop     ds
```

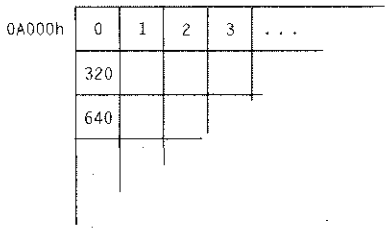
continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Masking is a useful feature that you can build into your icon display routines. Using masking, you are not bound to rectangular objects even if your data forms a rectangular block. An additional byte is needed to store the parts of the icon that are clear for every eight pixels. When masking is used, the byte-at-a-time display method is most effective: you can load the Bit Mask register (Port 3ceh, function 8) and then copy the four planes to the video RAM without having to modify the register. If you were using the plane-at-a-time method, you would have to modify the Bit Mask register for each byte displayed and calculate an offset into a table in order to retrieve these masks. **Display_Icon** uses the byte-at-a-time method with the masking added in (see Figure 2).

Figure 3 shows the layout for VGA mode 19 (13h), in which memory is laid out as one contiguous block with one byte per pixel. Because in this mode you don't need to worry about blocks being eight pixels, you can make the icons any size without affecting their layout design. In this mode I recommend laying out the icon pixel by pixel, from left to right and top to bottom. Instead of adding mask information to your icon, you can implement masking by selecting one of the 256 colors as your mask color. Because you are displaying a pixel at a time, don't display this color if you come across it. This will result in an instant mask.

FIGURE 3: Layout for VGA mode 19 (13h)

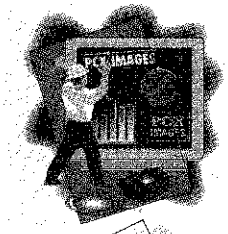


Mode 19 (13h) (320 bytes per line). One byte per pixel, one plane only. VGA only 320 x 200 with 256 colors.

The sample application generates fixed-size icons that are 16 pixels square: 2 bytes by 16. Each byte (eight pixels) needs a byte for each plane plus one byte for the Bit Mask. Thus, 2 bytes wide × 5 data each × 16 lines yields 160 bytes per icon. A predefined icon is included in the source to be used with the **Display_Icon** routine. The background is generated by a pixel pattern drawn in the shape of an icon used with the **Copy_Icon** routine. The displayed icon is also used as the source for **Copy_Icon** calls. The sample application displays a dithered background and then animates an icon across it.

Icons can open up the world of animation. Simply generate multiple icons of an image in different positions and then place one over the other in sequence. Or go a step beyond and redraw what was behind an icon before drawing the next. Most commercial products that feature animation use the latter technique in combination with page flipping and off-screen drawing to achieve the smooth-moving sequences that characterize them. PJ

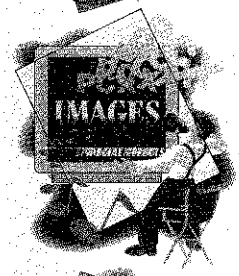
ADD THE POWER OF GRAPHICS TO YOUR PROGRAMS WITH GENUS



PCX PROGRAMMER'S TOOLKIT 4.0

Finally, a programmer's toolkit that helps you incorporate graphics into your programs quickly and easily. The PCX Programmer's Toolkit includes over 90 routines to display, save, scale, and print PCX bitmapped graphics from almost any program.

ONLY $249



GX EFFECTS 2.0* NEW!

Add spectacular special effects (F/X) to your programs, without being trapped in someone else's editor or slideshow program. Wipe, split, crush, slide, sand, drip, diagonal, spiral, random, or explode your graphics. Includes new animation support and sprite capabilities.

ONLY $199

* PCX Toolkit, GX Graphics, or GX Text recommended but not required.

GX TEXT 2.0 NEW!

Now you can display blazing bitmapped text in any graphics mode as simply as text mode. With GX Text, you can display your textual information right along with your graphical data, using the display or virtual buffers. Includes a graphical font editor and 1MB+ of fonts.

ONLY $149

GX GRAPHICS 1.0

A complete graphics library supporting all graphics primitives. Use the GX Graphics toolkit instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers – while supporting more video modes.

ONLY $199

TO ORDER ANY OR ALL OF THE GENUS GX DEVELOPMENT SERIES, CALL TOLL-FREE:

1-800-227-0918

VISA, MASTERCARD, AMERICAN EXPRESS, CHECK, C.O.D., PURCHASE ORDER

G·E·N·U·S MICROPROGRAMMING

2900 WILCREST • SUITE 145 • HOUSTON, TX 77042 • (713) 870-0737
FAX (713) 870-0288 • BBS (713) 266-9362

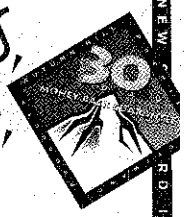
Circle 45 on Reader Service Card

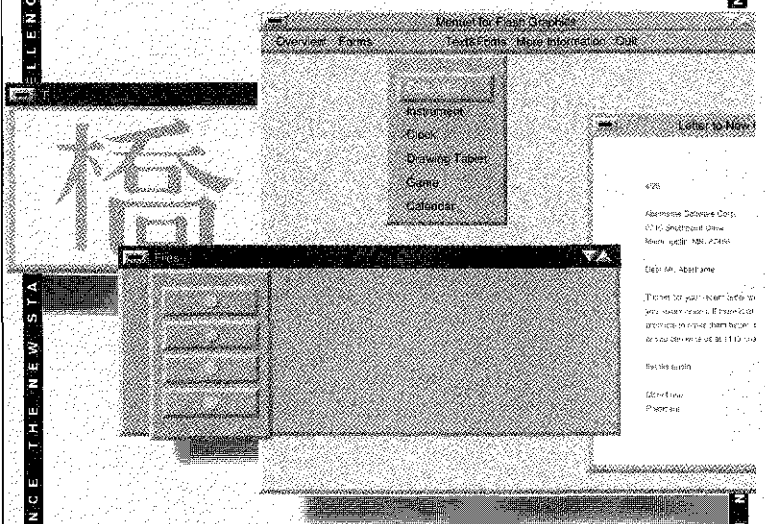
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE

Menuet™

The New Standard in Excellence in GUI Design for DOS, 386 Protected Mode, and UNIX!





Versions for MetaWINDOW, GSS*CGI, SCO*CGI, and Zortech Flash Graphics

Includes Interface Design Tool, Menu Design Tool, a Font Editor, and an Icon Editor

Over 1,000 pages of printed documentation in 2 volumes, plus full on-line documentation

Easy to use - we've had customers create demos in 3 days, finished applications in 3 weeks

Simply outstanding aesthetics - Emulate WINDOWS, MOTIF, or your own unique look!

DOS Version: **$325** UNIX Version: **$650**

Call for your free demo!

303.494.8865

AUTUMN HILL SOFTWARE

Circle **44** on Reader Service Card

1145 ITHACA DRIVE, BOULDER, CO, 80303, FAX: (303) 494-7802 BBS: (303) 494-8868

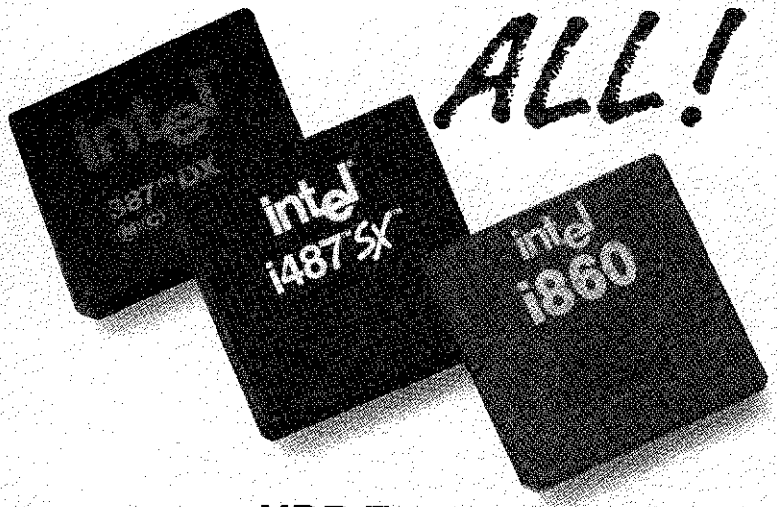
FORMERLY ITHACA STREET SOFTWARE. ALL TRADEMARKS ARE THE PROPERTIES OF THEIR RESPECTIVE OWNERS.

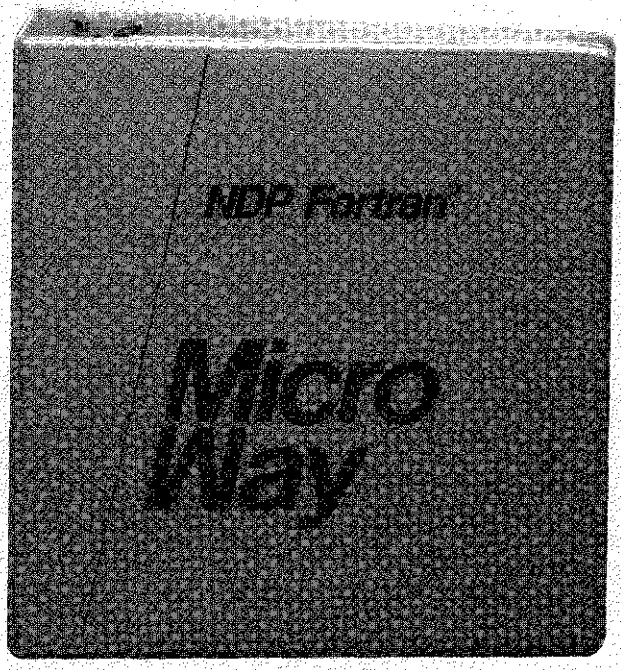
THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE

```
        pop     di
        pop     si
        pop     bp
        ret
_Copy_Icon endp

Video_ModeParms struc
                dw      ?       ; pushed BP
                dw      ?       ; return address pushed by call
    vm_mode     dw      ?       ; pass new video mode parameter
Video_ModeParms ends
_Video_Mode     proc    near    ; void Video_Mode(int Mode);
        push    bp
        mov     bp,sp           ; put sp in bp, to get parameters on stack
        mov     ax,word ptr [bp+vm_mode]
        xor     ah,ah           ; set video mode ( ah=0 )
        int     10h             ; call video BIOS
        pop     bp
        ret
_Video_Mode endp

Set_PixelParms          struc
                dw      ?       ; pushed BP
                dw      ?       ; return address pushed by call
    ps_xpos     dw      ?
    ps_ypos     dw      ?
    Color       dw      ?
Set_PixelParms          ends
_Set_Pixel proc near            ; void Set_Pixel(int x, int y, int Color)
        push    bp
        mov     bp,sp
        push    ds              ; save the data segment
        mov     ax,YOffset      ; get the video RAM address
        mov     ds,ax           ; set data segment to EGA/VGA
        mov     ax,[bp+ps_ypos] ; get the y coordinate
        mul     BytesPerLine    ; get the offset to line y
        mov     bx,[bp+ps_xpos] ; get the x coordinate
        mov     cx,bx           ;  and save in CX too
        shr     bx,1            ;  and divide by 8
        shr     bx,1
        shr     bx,1
        add     bx,ax           ; bx := BytesPerLine*y+(x/8)
                                ; we now have x,y memory offset
        and     cl,7            ; generate mask for one pixel in byte
        xor     cl,7
        mov     ch,1
        shl     ch,cl           ; ch := 1 >> (7-(x mod 8))

        mov     dx,3ceh         ; now select write mode 2
        mov     al,5
        out     dx,al           ; select mode (register 5)
        mov     dx,3cfh
        mov     al,2
        out     dx,al           ; and set to write mode 2

        mov     dx,3ceh         ; set up mask
        mov     al,8
        out     dx,al           ; select mask (register 8)
        mov     dx,3cfh
        mov     al,ch
        out     dx,al           ; set mask

        mov     al,[bx]         ; read from card and color dot
        mov     ax,[bp+Color]
        mov     [bx],al         ; write color to dot

        mov     dx,3ceh         ; restore default mode
        mov     al,5
        out     dx,al           ; select mode (register 5)
        mov     dx,3cfh
        mov     al,0
        out     dx,al           ; set write mode back to 0
        mov     dx,3ceh
        mov     al,8
        out     dx,al           ; select mask (register 8)
        mov     dx,3cfh
        mov     al,0ffh
        out     dx,al           ; and set default mask
        pop     ds              ; restore data segment
        pop     bp
        ret
_Set_Pixel      endp

_TEXT ends
end
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Only ONE Fortran Drives Them ALL!



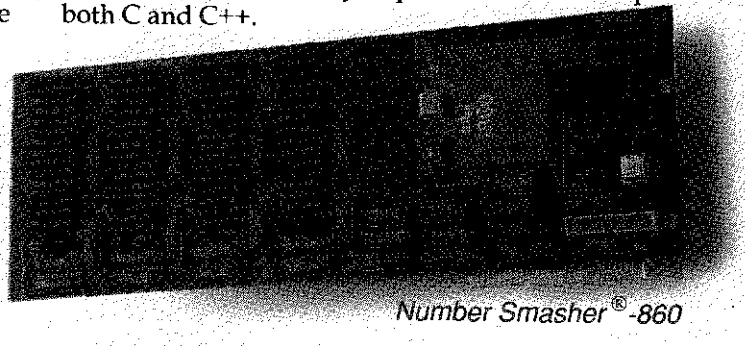


NDP Fortran delivers mainframe performance driving the Intel 387™, 487™SX or i860™.

Since 1982, Microway has provided numeric coprocessor support for scientists and engineers. This support has reached a pinnacle with NDP Fortran, the only PC Fortran available which efficiently drives all Intel advanced coprocessors, plus the i860 RISC, single chip Super Computer.

Whether you are running DOS, UNIX, DESQview, Windows, XENIX, or SunOS, there is a version of NDP Fortran which will give you globally optimized access to the four gigabyte address space of the 386, 486 or i860. Code produced for their 32-bit flat model accesses large data structures and arrays a factor of four faster than 16-bit products which depend on the huge model or EMS memory.

Because Microway's NDP compilers use a common back end for Fortran, Pascal, C and C++, you can access code generation features from NDP Fortran that until recently were thought to benefit C users only. For example, the inlining of small procedures, which is the hottest optimization in both C and C++, was not considered to benefit Fortran users. However, inlining works for all modular code. When inlining was applied to the Whetstone running on our Number Smasher-860, the benchmark jumped from 28.6 to 68.5 MegaWhetstones! Similarly, loop unrolling, which has the reputation for being a mainframe Fortran optimization only, dramatically improves the inner loops of both C and C++.



Number Smasher® -860

NDP Fortran is a full ANSI 77 that is 99% compatible with VMS Fortran and has extensions from BSD, DOD and Microsoft. The DOS version includes the GREX™ library of 88 graphics primitives and I/O functions that give it a BASIC-like flavor.

Finally, every Microway product comes with the best technical support in the industry. Call 508-746-7341 for your free copy of the *Journal of Numeric Data Processing* to see how we used inlining to speed up the Whetstone, and loop unrolling to speed up Livermore Loops.

i860, 387, 487SX and Intel are trademarks of Intel Corp.
Microway and Number Smasher are registered trademarks of Microway, Inc.

Microway®

Contact this advertiser directly.

The World Leader in PC Numerics

Corporate Headquarters, Box 79, Kingston, MA 02364 USA • TEL 508-746-7341 • FAX 508-746-4678
U.K./Europe 081-541-5466 • France 01 43 2 69593 • Germany 069-75-2023 • Holland 40 836455 • Italy 02-74.90.749 • Norway 9 876656

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Integrate Divide and conquer

Examining math coprocessors for the 80386 and 80486

By Mark J. Barrenechea

Historically, floating-point arithmetic has been slower than similar integer-based arithmetic. That's why designers have used numeric processing units as separate execution devices. Floating-point comparisons, for example, can be expensive: on an 80386, it is faster to do floating-point comparisons in the integer unit than in the floating-point unit. But as the 80x86 has evolved, so has the coprocessor, and the performance gap between the floating-point and integer units has narrowed so much that the differences are negligible. This narrowing has been caused by advances in silicon technology, improved microcode, and the increased amount of software logic contained in hardware.

INTEGRATION IS THE WAY

The emerging trend of the nineties is to integrate the traditional integer and numerics units on a single chip. The increasingly crowded and clouded numerics market currently includes the i486DX, SPARC, MIPS R2000/3000/4000, RS6000, and i860 chips. Unlike their pedantic, clunky predecessor, the Intel 80386, these newer chip designs have highly coupled execution units.

For the Intel 80386, the math coprocessor market is straightforward and well defined. The best-known coprocessors are the Intel 387, the Cyrix EMC87, the Cyrix 83D87, and the Weitek 3167. (Two lesser-known products are not reviewed here, but vendor information is provided below.) Two coprocessors are available for the 80486: the Weitek 4167 and the Intel 487 (which is a 387 bundled with the 486). The 387 is not bundled with the 486SX.

This article explores the chip technology and performance of the "divide and conquer" approach taken by the Intel 80386/80486 and its math coprocessors. Each math coprocessor will be discussed in terms of its design issues and features, and benchmarks will be run to compare like results.

A MATTER OF MAPPING

There are two physical approaches to attaching a coprocessor inside a PC: I/O mapping and memory mapping. The Intel 387/487 and Cyrix 83D87 are I/O mapped. The Weitek 3167/4167 is memory mapped. The Cyrix EMC87 works in either mode, but Cyrix recommends that modes not be mixed.

In I/O mapping, the coprocessor is linked to the 80386 by a direct connection over the I/O bus. A memory-mapped coprocessor uses a memory address range to communicate with the 80386.

I/O-mapped math coprocessors are directly connected to the 80386. (Another term for this class of coprocessor is *387-compatibility mode.*) The coprocessor recognizes any instruction that is prefaced by an appropriate ESCAPE sequence; no memory addresses are set aside for special purposes such as instruction encoding. I/O-mapped coprocessors use I/O port addresses that are inaccessible to the programmer. These 32-bit addresses have their high-order bit set (0x80000000) [Reference 1].

Because the 80386 CPU and its math coprocessor have separate execution units, in theory the two units can "coprocess"—that is, execute instructions in parallel. The drawback is that handshaking and synchronization cycle delays can offset the gains made by scheduling. Instruction scheduling is an underused optimization for I/O-mapped math coprocessors (see my article, "Peak performance: On to the 80486," *PJ* 8.6).

Mark J. Barrenechea is vice-president of language development at MicroWay, Inc., in Plymouth, Massachusetts.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Intel has defined the I/O-mapped architecture well, as evidenced by the number of alternative 387-compatible math coprocessors and floating-point emulation libraries that exist today.

AN EXPLANATION IS DUE

Memory-mapped coprocessors require more explanation because of their clever design. The *Weitek WTL1167 Software Design Guide* [2] provides a good overview of how memory-mapped coprocessors work. Some of that information follows.

When the 80386 reads from memory, it emits signals on its control output pins, along with the actual memory address on the address pins. Figure 1 shows how an 80386 and a memory device interact: the device uses the upper bits of the address to determine whether or not to activate. When it does activate, the middle and lower address bits are used to determine which memory cells are accessed. The device responds by sending the value of a memory cell over the 80386 data bus, where it can be accessed by the 80386.

The 80386 CPU communicates with memory devices via this message passing. This device does not have to be a memory device, however; it can be a coprocessor that is activated when a certain address range is accessed (read or written). At the hardware level, a memory-mapped coprocessor masquerades as a memory device. At the software level, it operates as a math coprocesser (see Figure 2).

The numbers show that memory-mapped coprocessors do, in fact, outperform I/O-mapped coprocessors.

Both the Weitek 3167/4167 and the Cyrix EMC87 require a 121-pin extended math coprocessor (EMC) socket on the 80386 motherboard. The larger sockets are necessary to accommodate these memory address lines for the memory-mapped coprocessors.

Even though memory-mapped coprocessors cannot coprocess with the 80386, in theory they should execute faster because they communicate via memory, not the I/O bus. I/O-mapped coprocessors move data and instructions over multiple cycles, whereas information can be exchanged in one cycle (a write to memory) for memory-mapped coprocessors. The numbers show that memory-mapped coprocessors do, in fact, outperform I/O-mapped coprocessors.

Memory-mapped math coprocessors do not have well-defined interfaces. Each chip manufacturer can design its own address ranges, instruction encoding, and structural details. The lack of a uniform interface is illustrated by the assembly code that is necessary to drive the Cyrix EMC and Weitek 3167/4167 (see below).

FIGURE 1: Interaction of the 80386 and a memory device

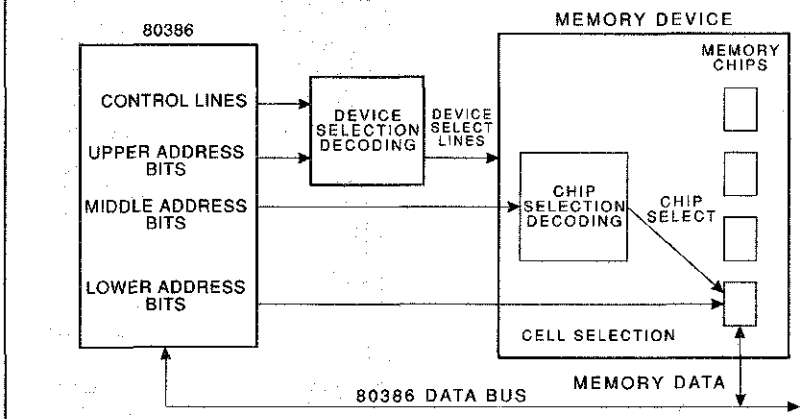
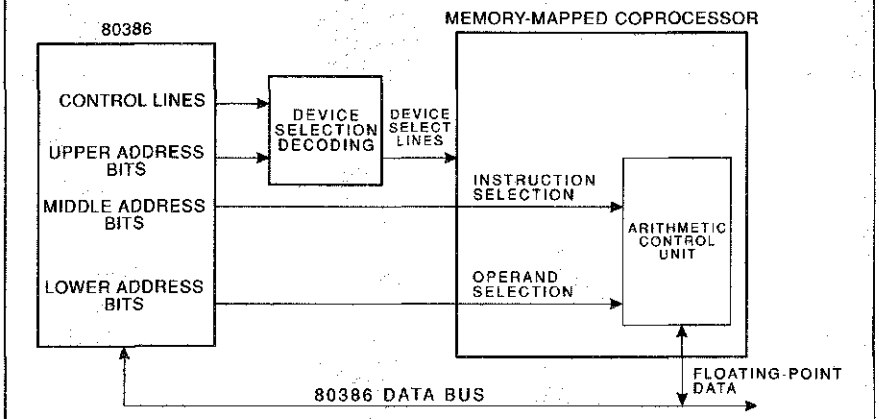


FIGURE 2: The 80386 with a memory-mapped coprocessor



THE PLAYERS

The Weitek 3167/4167 provides a register set of 32 32-bit single-precision registers. These can be doubled up, making 16 64-bit double precision registers. The use of register R0 is restricted, but this doesn't affect the application programmer. Unlike the Intel 387/487 and Cyrix 83D87, the 3167/4167 does not support 80-bit extended reals.

The Weitek coprocessors have traded accuracy for speed. As with the i860, certain features of the IEEE standard are not implemented. Additionally, denormal arithmetic is not performed; all denormals are flushed to zero.

The Weitek 3167/4167 features an interesting instruction set. One instruction, the multiply accumulate, can

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

be used to generate high-speed matrix manipulation:

```
wfmac     ws17, ws5 ; ws2 = (a11 * x) + (a21 * y)
wfmac     ws18, ws6 ; ws2 = (a11 * x)
          + (a21 * y) + (a31 * Z)
wfmac     ws19, ws7 ; ws2 = (a11 * x)
          + (a21 * y) + (a31 * Z) + (a41 * w)
```

This instruction set has important implications when one of the benchmarks is applied.

Consider the following C program:

```
extern float f1;
extern float f2;
float foo(voidv ){
  return ((f1 + f2) * (f1 - f2));
  }
```

Listing 1 shows the generated Weitek 3167/4167 assembly language code. Note that the Weitek macros (**wfld, wfadd, wfneg,** etc.) and 386 instructions are provided. The memory move instructions that are typical of a memory-mapped coprocessor,

```
; wfld ws4,dword ptr ds:_f1
  mov  eax,dword ptr ds:_f2
  mov  dword ptr ds:[000000010H],eax
```

access a range of memory that activates the Weitek to load a single-precision register, **ws4,** with the value of **f1.**

From a functional standpoint, the Cyrix EMC is 100% compatible with the 387. It has only eight registers and an internal 80-bit register format. If a compiler already generates 387 code, the Cyrix EMC intercepts the 387 instructions and substitutes the EMC memory-mapped equivalent. A single 387 instruction can (and usually does) generate multiple EMC instructions. The EMC code in Listing 2 is functionally equivalent to the C program given above.

The Intel 387 is a stack-based math coprocessor whose instruction set is currently popular. The 387 instructions generated from the C program are provided in Listing 3.

THE BENCHMARKS

Because a coprocessor may have special high-speed instruction sets for handling a particular problem (transcendentals, matrix operations, scalars, multiply accumulates, 3-D rendering, etc.), a good floating-point benchmark must allow coprocessors to be tested in real-world applications. The WhetStone benchmark is not perfect, but it has a reasonable mix for testing math coprocessors that work with the 80386 and 80486. It contains scalar floating-point code, transcendental code, and matrix operations.

TABLE 1: Benchmark results, single precision (KWhets/sec)

Coprocessor	WhetMat	WhetScal	WhetTrans	WhetStone
386:				
Cyrix 83D87	851	1445	2217	5075
Cyrix EMC87	1848	4050	3557	7934
Weitek 3167	2699	4332	1582	6719
Intel 387DX	849	1492	910	3800
486:				
Weitek 4167	5967	12995	4238	15148
Intel 487DX	2460	3876	1610	7934

TABLE 2: Benchmark results, double precision (KWhets/sec)

Coprocessor	WhetMat	WhetScal	WhetTrans	WhetStone
386:				
Cyrix 83D87	711	1389	2890	4673
Cyrix EMC87	1186	3570	4427	6757
Weitek 3167	1120	3804	1570	5378
Intel 387DX	713	1339	1039	3509
486:				
Weitek 4167	2783	10756	3951	12995
Intel 487DX	2263	3651	1734	7577

MicroWay has broken out the standard WhetStone into three separate benchmarks: WhetMat (the matrix operation), WhetScal (the set of floating-point scalar operations), and WhetTrans (the transcendentals). Listing 4 shows the subroutines for each benchmark. Each piece was run in single and double precision.

The WhetStone benchmark is not perfect, but it has a reasonable mix for testing math coprocessors that work with the 80386 and 80486.

The WhetStone benchmark can also be used to test parameter-passing speeds. On a RISC device, this test can be significant. It is not significant on modern CISC architectures, where function calling is negligible and a good function in-liner can remove the call altogether.

When it is "rating" a coprocessor, a benchmark can represent two aspects: the coprocessor's performance in a given floating-point arena and the compiler's ability to recognize and generate optimal code sequences. If a compiler cannot generate matrix operations for a given coprocessor, the matrix instructions are unused or untested, thus underrating the coprocessor's ability. This was taken into consideration when these benchmarks were run.

TEST RESULTS

The 80386 results were generated on an AIR 386AS motherboard, with 64K of cache, using an Intel 80386 33-MHz processor and 8MB of main memory. All coprocessors were 33 MHz. NDP-386-FORTRAN 3.2.0 was used.

The 80486 results were generated on a MicroWay NumberSmasher 486-ISA burst-mode motherboard with no cache. All other conditions were the same.

MicroWay NDP-FORTRAN can generate highly tuned code for each of these benchmarks and coprocessors. Thus, the numbers in Tables 1 and 2 represent an accurate measure of apples to apples—a solid mix of compiler ability and coprocessor features (this benchmark could have been written in C, C++, or Pascal), and, if transliterated

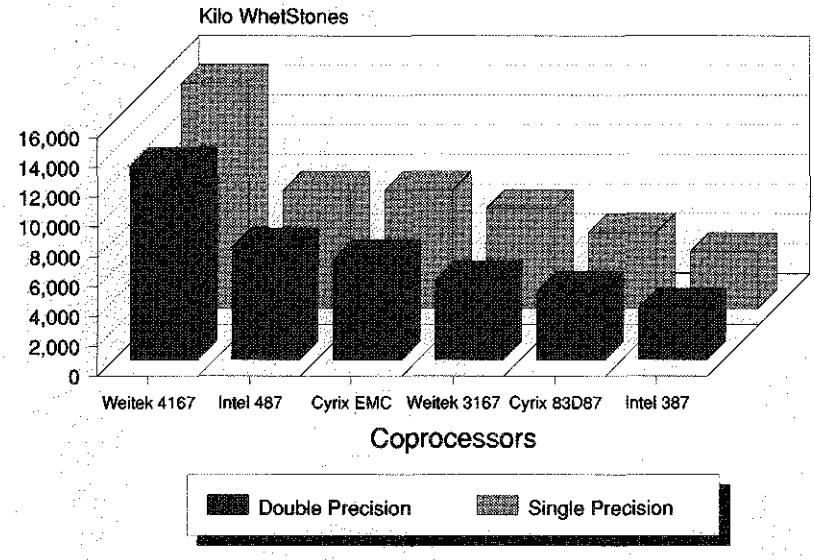
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

correctly, the numbers that were generated would have been the same as the Fortran equivalent. The two tables show the single- and double-precision results, respectively. The best rating in each category is highlighted. Figure 3 presents the overall WhetStone rating and compares single- and double-precision results.

It is easy to see that the competition for top chip is between Weitek and Cyrix. The Intel 387 and 487 perform relatively slowly in each of their respective categories. Any discussion centered on performance must exclude them.

On the 80386 in single precision, the Weitek wins in WhetMat (matrix operations), because it has a special high-speed multiply accumulate for matrix operations. The Weitek and Cyrix EMC are fairly close in WhetScal (scalar arithmetic), although the Weitek wins. In WhetTrans, the Cyrix EMC is the clear winner; it's highly tuned with specialized microcode to do very fast

FIGURE 3: Coprocessors for the 80386/80486



640K WINDOWS ROM LIM EMS HMA DOS 5.0 XMS BUFFERS UMB 386MAX 486SX DRIVERS 386SX NETWORKS TSRs PAGEFRAME IPX

RAM MEMORY BIOS CACHE VGA MOUSE 386 BLUEMAX 486 HIGH INITIALIZE

Only the most intelligent memory managers can pass this test.

It takes intelligence to get maximum memory for your 386 system. But whose would you rather use—yours or your memory manager's? Here's a little quiz to help you make the smart choice.

1 True or False—All memory managers are alike.

False. Most memory managers free up space for applications by moving TSRs and device drivers from conventional memory into high DOS memory. With other products, you could spend days guessing about the best way to do that. But MAX uses its intelligence to find the best possible fit—automatically. Guaranteed. And because many TSRs need much more space to load than they need to run, MAX has an exclusive feature that "borrows" up to 64K of high DOS for loading, so it can pack even more in. With MAX, you always get the most memory possible.

2 True or False—Using TSRs in Windows is a great way to crash your system.

With other products, that's definitely true. But thanks to another MAX exclusive, you can use nearly any pop-up utility as many times as you want.

3 With DOS 5.0, there's no need for a memory manager—right or wrong?

That depends. DOS 5 does free up some memory with its smaller program size. And it does let you place programs into high DOS manually. But for optimal memory management, you need MAX—the powerful, automated, full-service program that specializes in memory. So while DOS 5 is good, DOS 5 with MAX is outstanding.

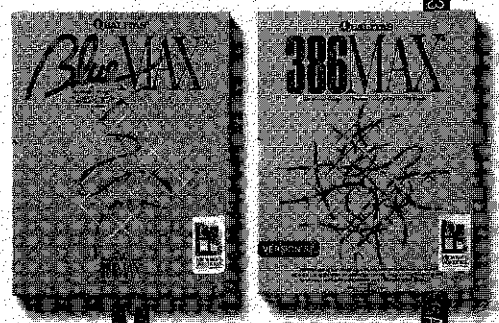
The answer is easy. The delivery is free.

Why not graduate to the Intelligent Memory Managers? There's 386MAX® for any IBM compatible 386 or 486 based system at just $130. And BlueMAX™ at $155, the only way to get 250% more high DOS memory on all 386 and 486 IBM PS/2s.

Order direct before 3:00 pm Eastern Time, and overnight shipping is free.* We'll even send a free copy of ASQ,™ the intelligent memory analyzer.

See your dealer or call toll-free today.

1-800-676-6386 ext. 824



The Intelligent Memory Managers™

640K WINDOWS ROM LIM EMS HMA DOS 5.0 XMS BUFFERS UMB 386MAX 486SX DRIVERS 386SX NETWORKS TSRs PAGEFRAME IPX

RAM MEMORY BIOS CACHE VGA MOUSE 386 BLUEMAX 486 HIGH INITIALIZE

© 1991 Qualitas. Qualitas, 7101 Wisconsin Avenue, Suite 1386, Bethesda, MD 20814. *All company and product names are trademarks or registered trademarks of their respective owners.* **System Requirements: Any 386 or 486 PC or PS/2, min. 256KB of extended memory, DOS 3.0 or higher, high density floppy disk or hard disk drive.**

QUALITAS®

**Offer valid in North America only.*

Circle 48 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
_foo    proc    near

        mov     eax,dword ptr ds:_f1
        mov     dword ptr ds:[000000410H],eax

;       wfld    ws4,dword ptr ds:_f1
        mov     eax,dword ptr ds:_f2
        mov     dword ptr ds:[000000010H],eax

;       wfadd   ws4,dword ptr ds:_f2
        mov     eax,dword ptr ds:_f2
        mov     dword ptr ds:[000002808H],eax

;       wfneg   ws2,dword ptr ds:_f2
        mov     eax,dword ptr ds:_f1
        mov     dword ptr ds:[000000008H],eax

;       wfadd   ws2,dword ptr ds:_f1
        mov     byte ptr ds:[000000888H],al
;       wfmul   ws2,ws4
        ret
        align   4

_foo    endp
        extrn   _f2:dword
        extrn   _f1:dword
        public  _foo
        end
```

LISTING 2

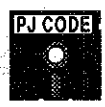
```
_foo    proc    near
        push    ebp
        mov     ebp,esp
        sub     esp,16
;       fld     dword ptr ds:_f2
        mov     eax,dword ptr ds:_f2
        mov     dword ptr ds:[512],eax
;       fsubr   dword ptr ds:_f1
        mov     eax,dword ptr ds:_f1
        mov     dword ptr ds:[160],eax
;       fst     dword ptr [ebp]+(-16)
        mov     eax,dword ptr ds:[576]
        mov     dword ptr [ebp]+(-16),eax
;       fld     dword ptr ds:_f2
        mov     eax,dword ptr ds:_f2
        mov     dword ptr ds:[512],eax
;       fadd    dword ptr ds:_f1
        mov     eax,dword ptr ds:_f1
        mov     dword ptr ds:[0],eax
;       fst     dword ptr [ebp]+(-12)
        mov     eax,dword ptr ds:[576]
        mov     dword ptr [ebp]+(-12),eax
;       fmulp   st(1),st
        mov     dword ptr ds:[3364],eax

;       fstp    dword ptr [ebp]+(-8)
        mov     eax,dword ptr ds:[608]
        mov     dword ptr [ebp]+(-8),eax
        mov     edx,[ebp]+(-8)
        mov     [ebp]+(-4),edx

;       fld     dword ptr [ebp]+(-4)
        mov     eax,dword ptr [ebp]+(-4)
        mov     dword ptr ds:[512],eax
        leave
        ret

        align   4

_foo    endp

        extrn   _f2:dword
        extrn   _f1:dword
        public  _foo

        end
```

LISTING 3

```
_foo    proc    near
        push    ebp
        mov     ebp,esp
        sub     esp,16

;       .bf:    ;
        fld     dword ptr ds:_f2
        fsubr   dword ptr ds:_f1
        fst     dword ptr [ebp]+(-16)
        fld     dword ptr ds:_f2
        fadd    dword ptr ds:_f1
        fst     dword ptr [ebp]+(-12)
        fmulp   st(1),st
        fstp    dword ptr [ebp]+(-8)
        mov     edx,[ebp]+(-8)
        mov     [ebp]+(-4),edx

;       .ef:    ;
        fld     dword ptr [ebp]+(-4)

       leave
        ret

        align   4

_foo    endp

        extrn   _f2:dword
        extrn   _f1:dword
        public  _foo

        end
```

LISTING 4

```
      SUBROUTINE WHETMAT
      DIMENSION  A(140,140), B(140,140), C(140,140)
C        ...
      DO 5 I = 1, 100
         DO 5 J = 1, 100
            A(I,J) = 50 * I + J
            B(I,J) = 77/J + 11/I
            C(I,J) = 0.0
    5 CONTINUE
      END



      SUBROUTINE WHETTRANS
      DO 78 I=1, N7, 1
         X = T*ATAN(T2*SIN(X)*COS(X)/(COS(X+Y)+COS(X-Y)-1.0))
         Y = T*ATAN(T2*SIN(Y)*COS(Y)/(COS(X+Y)+COS(X-Y)-1.0))
   78 CONTINUE
C        ...
      DO 118 I=1,N11,1
         X = SQRT(EXP(LOG(X)/T1))
  118 CONTINUE
      END


      SUBROUTINE WHETSCAL
      X1 = 1.0
      X2 = -1.0
      X3 = -1.0
      X4 = -1.0
C        ...
      DO 18 I =1, N1, 1
         X1 = (X1+X2+X3-X4)*T
         X2 = (X1+X2-X3+X4)*T
         X4 = (-X1+X2+X3+X4)*T
         X3 = (X1-X2+X3+X4)*T
   18 CONTINUE
      END
```



All code in this issue is available through our bulletin board services or directly from PJ on disk. For information see page 81.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

transcendentals. The overall single-precision winner on the 80386 is the Cyrix EMC, by 18%.

On the 80386 in double precision, the Weitek is in catch-up mode. The Cyrix EMC wins the WhetTrans, the WhetMat, and the overall WhetStone. Even more impressive than the EMC's winning both the single- and double-precision benchmarks is its ability to perform better in double-precision transcendentals than single-precision by 24%. This means programmers get greater precision and accuracy in less time. Cyrix obviously did its homework in this area.

On the 80486, the Weitek 4167 beats the Intel 487 in *all* single- and double-precision tests. The Weitek 4167 does, however, lose to the Cyrix 386 EMC in double-precision transcendentals by 12%. Again, the Cyrix EMC transcendentals fly. On the 80486, the Weitek 4167 is the clear winner.

It's interesting to observe that a 386 with an EMC87 is competitive with an Intel 486/487 on scalar code and far superior on transcendentals.

CONCLUSION

If you wanted a math coprocessor for the 80386 three years ago, you had two choices: an Intel 80387 or a Weitek 3167. Today, you have more than six choices. In a year, you will have at least eight—Chips & Technologies and AMD will be entering this already crowded market. Wall Street analysts say that the 387DX is the highest-profit-margin chip ever produced by Intel, and the other manufacturers are obviously trying to grab a share of this profit center. The 80387 has carried a retail price of almost 50 times the manufactured cost of about $20. The 80486 choices (the Intel 487 and the Weitek 4167) are where the 80386 choices were three years ago.

The competition among chips can be profitable for some and deadly for others. As the math coprocessor industry evolves, chip manufacturers are subject to natural selection. When I began this article, the Intel 387DX-33 was selling for more than $500. It has now dropped to less than $200. (Because of such fluctuations, prices are not listed with the product information.) It's hard to predict which coprocessors will emerge as the fittest.

The fastest math coprocessor at any given moment is king of the hill. Currently, the Cyrix EMC is enjoying this reign on the 80386, and the Weitek 4167 on the 80486. But watch out—no one knows where the chips may fall.

REFERENCES

1. Intel Corporation (1987). *80387 Programmer's Reference Manual*, pp. 2-6.
2. Weitek Corporation (1987). *Weitek WTL1167 Software Design Guide.*

VENDOR INFORMATION

Cyrix EMC87 and Cyrix 83D87
Cyrix Corporation
2703 North Central Expressway
Richardson, TX 75080
(800) 327-6284, (214) 234-8387
Circle **214** on Reader Service Card

80387DX and 80487DX
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95052
(916) 351-5574, (800) 538-3373
Circle **215** on Reader Service Card

Weitek 3167 and Weitek 4167
Weitek Corporation
1060 East Arques Avenue
Sunnyvale, CA 94086
(408) 738-8400
Circle **216** on Reader Service Card

IIT 3C87
Integrated Information Technology, Inc.
2445 Mission College Boulevard
Santa Clara, CA 95054
(408) 727-1885
Circle **217** on Reader Service Card

US83C87
Specialty Development Corporation (SDC)
1001 Capital of Texas Highway
South Building 1
Austin, TX 78746
(512) 327-8606
Circle **218** on Reader Service Card

People are talking about us.

F77L-EM/32 & Lahey Ergo OS/386

Port mainframe programs as large as 96MB to 386/486's with this 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 4.0 includes: Programming Tools, Popular Fortran 90 features, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $995 OS/386 $395

F77L

The fastest real-mode compiler available. F77L can take advantage of your 386 PC by generating 32-bit instructions. New Version 5.0 includes: Fortran 90 features, Weitek support, and Video Graphics. $595

Lahey Personal Fortran 77

Version 3.0: Full ANSI 77, Editor, Debugger, Linker, Library Manager, Microsoft and Borland C interfaces. A great learning tool at an unbeatable price. $99

Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123

FORTRAN IS OUR FORTE

Circle **49** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Window layers

Part 3: How a large window and a small viewport can work together

By Allen Holub

This article describes the high-level part of a windowing function library that combines the text-buffer and viewport manipulation routines from two previous articles (*PJ* 9.1, 9.4) into a useful windowing system. (The low-level component will be covered in an upcoming issue.) Windows can be created, destroyed, hidden, and moved in the usual way; but a window can also be large, with only a small viewport into the underlying text displayed on the screen. Moving the window cursor to an area of the undisplayed underlying text causes the text to slide under the viewport until the appropriate text comes into view.

OUT FOR A VIDEO SCROLL

Before starting in earnest, I need to make a few minor corrections to the previous articles. In my last article (9.4), I forgot to describe and include the code for one of the viewport functions, which is included on this issue's code disk:

```
int v_scroll_region( viewport *v,
                     int up, int to_left,
                     int left, int top, int right,
                     int bottom )
```

This function scrolls the text in the viewport horizontally or vertically (or both), but only that text bounded by the box given in the four last arguments is scrolled. (These are window-relative row and column numbers.) The text is moved up by **up** lines, and it's moved **to_left** lines to the left. The opened-up regions are filled with spaces. The **up** and **to_left** arguments may be zero, and they may also be negative. The text moves to the right if **to_left** is negative; it moves down if **up** is negative. The cursor position is not changed by the scroll. This function may not be used to clear the viewport—if the requested movement is greater than the viewport dimension, **v_scroll_region** returns false (and no movement occurs along that axis). Use **v_clear** to clear a viewport. The function also returns false if the bounding box is outside the window. (In such cases it truncates the size of the bounding box to fit).

The **v_scroll** function that was included last time is actually a macro in VIEWPORT.H that looks like this:

```
#define v_scroll(v,u,l) v_scroll_region  \
        ((v),(u),(l),0,0, VMAXCOLS,VMAXLINES)
```

I also found a few bugs and omissions in the previous code. In TEXTBUF.H (Listing 1, *PJ* 9.1) you need a backup macro to complement **b_advance(b)**:

```
#define b_backup(b) ( (b)->col--, (b)->p-- )
```

This macro backs up the cursor one notch. I've also changed the name of the **b_setrc** macro to **b_gotorc** so that its name agrees with the **v_gotorc** macro used for viewport management.

In addition to the **b_clearline** function that I described in *PJ* 9.1, it turned out to be handy to have a **b_cleartoeol,** which clears from the cursor to end of line. It looks like this:

```
#define b_cleartoeol(b,c) \
    bufset((b)->p, (c), (b)->ncols - (b)->col)
```

This function matches the viewport's **v_cleartoeol(v)** function (in V_CLEAR.C on the code disk).

I've added another function to the buffer-management items (shown in Listing 1). This function, **b_last(textbuf *b)**, returns the column number of the rightmost nonwhite column on the current line.

Finally, the **b_delete** macro (from 9.1's Listing 1) had a bug in it. The new version is as follows:

```
#define b_delete_c(b,n,c) (  bufmove( (b)->p, \
        (b)->p+(n), (b)->ncols -((b)->col+(n))), \
                bufset ( ( (b)->p - (b)->col) + \
                ( (b)->ncols - (n) ), (c), (n)) )
```

AN ARCHITECTURE OF LAYERS

Let's take a look at the system within which windows function. Three layers are involved in any given window. At

---

Allen Holub is a teacher, writer, consultant, and composer. He teaches C and compiler design for the U.C. Berkeley Extension. His most recent book is Compiler Design in C. *Allen can be reached at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705.*

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the top is the viewport—the portion of the window that's displayed on the screen (viewports are covered in *PJ* 9.4). This viewport is a window, so to speak, into a larger text buffer that resides in memory. The text buffer is, in turn, a window into a much larger underlying file. That is, the text buffer holds some limited number of lines from the file, and the viewport holds some limited number of lines and columns from the text buffer. The buffer and the file are the same width—the maximum line width is the same in both layers. The viewport can be narrower than the buffer, but not wider.

A provision is also made for adding a built-in editor to the window. The functions, as presented here, are all low-level: move the cursor, print a character, and so forth. Ideally, these functions could be buried inside an "editor" function that would make the actual window calls. Looked at in an object-oriented fashion, the user interface would get a stream of editor directives from the input system. Some of these (such as standard characters) would represent an "insert a character here" directive to the editor. Other characters (such as function keys) would do cut operations, move the cursor long distances, and so forth. This stream of characters can be thought of as a string of "messages" that are passed to an editor object. The editor would then call the low-level window functions to do the actual work.

The viewport is a window into a larger text buffer in memory. The text buffer is a window into a larger underlying file.

The editor object, however, needs a structure of some sort to store its own internal state. It can't use global variables for this purpose, because several windows, each with its own editor, might be active. Similarly, the window functions need a structure to keep track of their states (current cursor position, relative position of the viewport with reference to the underlying text, and so forth). You could pass two pointers to the editor functions, which would then pass the window-structure-pointer back to the windowing workhorse functions. But it's easier to pass just one pointer.

You could accomplish this in C++ by creating an editor class and then deriving the window class from it:

```
class editor
{   // Editor state data
    // and functions
}
class window : public editor
{   // Window state data
    // and functions
}
```

You could then pass a pointer to a **window** through to an **editor,** and everything would work without difficulty.

In C, you have to simulate derivation by nesting the editor structure inside the window structure, like this:

```
struct editor
{
    /* editor state */
};
struct win
{
    struct editor e;
    /* window state */
};
```

Because the **editor** is at the top of the **win**, a pointer to a **win** structure will also point at the nested **editor** structure. (I use **win** instead of **window** because Borland has used that word for something else.)

WORKHORSE FUNCTIONS

To make the system as general-purpose as possible, the editor defines a set of functions that it needs to do its work—functions that move the cursor, print characters, delete characters, and so forth. This editor can then be used in virtually any environment simply by providing the required functions. The lowest-level loop would look like this:

```
make_editable_window
{
    win *w;
    w = open_window;
    while( getchar != EOF )
        editor( c, w );
    close_window;
}
```

The **w** pointer is passed to the editor function, which treats it as if it were a pointer to an **editor** structure. This allows you to have several windows, each with its own editor. The editor function calls the window workhorse functions. Because the **editor** pointer is also a **win** pointer, the editor can pass the incoming pointer back down to the workhorse functions, which manipulate the **win** component of the structure. The window functions don't manipulate the **editor** component, and the editor functions don't manipulate the **win** component.

These workhorse functions can be used independently, of course. The editor object is a good conceptual tool, but your program could just as easily call the low-level window functions directly. Nonetheless, I've designed hooks for the editor into the current functions and data structures so that an editor can later be added easily; I'll discuss this in detail next time.

GETTING MORE CONCRETE

The definitions for this system are split into two files. The first, WINDOW.H (Listing 2), holds definitions that are used to manipulate windows at a higher level. The **win** structure represents the window. It starts with an **editor**, as promised. The editor definition is in the header file TOOLS/EDITOR.H (available on this issue's code disk), which will be discussed next time. The preceding **#define ETYPE struct win** customizes the definitions in EDITOR.H to the current application. EDITOR.H contains function prototypes for the low-level functions that the editor needs to do its work, and these functions are passed an **ETYP** first argument in the prototype:

```
int  e_open( ETYP *const p );
```

The real functions, however, are declared with a **win** first argument, like this:

```
int  e_open( win *const p ){...}
```

The macro maps **ETYP** into the correct type.

The next fields are the **viewport** and **textbuf** for the window. The **row_off** and **col_off** fields hold the distance from the upper-left corner of the text buffer to the upper-left corner of the viewport, thus positioning the viewport with respect to the text buffer. **file_off** does the same thing with the text buffer and the underlying file. It's the distance (in lines) from the top line of the file to the top line of the text buffer. The **bottom** field holds the line number of the bottom

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

line of the file. The cursor position is not permitted to go past this position.

The various **update_** fields are used to disable viewport updating. This is handy when, for example, you want to write to a hidden window. These fields hold the offsets and cursor position that were current when updating was disabled. The cursor is restored to this position when updating is reenabled. An application program can modify or search around in the underlying buffer to its heart's content, and the old position will be restored when updating is reenabled.

The **update** and **dirty** fields are also used here: **update** is false when viewport updating is disabled. The **dirty** bit is set true if the text buffer has been modified, rather than just examined, since updating was disabled. If the buffer is clean, this preempts an unnecessary screen refresh. **engine_active** is another hook for editing-engine support. It's set true after the engine is initialized.

FOUR FILE MANIPULATORS

The four function pointers at the bottom of the structure take care of underlying-file manipulation. If they're NULL, then there is no underlying file. Otherwise, they work as follows. The

```
char *(*import)(
        struct win *this, int lnum );
```

function is used to get lines from the underlying file. It is passed a line number and returns a string that holds that line from the file. The

```
int   (*insert)(
        struct win *this, int lnum );
```

function inserts a blank line at the indicated position in the file. The original contents of that line, and all lines beneath it, are moved down one notch. The

```
void  (*export)(
        struct win *this, int lnum, char *line,
        int nc, int len );
```

function is the opposite of **import**. It is used to copy a modified line from the text buffer to the underlying file and is passed the number of the line to overwrite, the contents (**line**), the size of the line buffer (**nc**), and the number of characters actually used in the **line** string (**len**).

The function

```
void  (*remove)(
        struct win *this, int lnum,
        char *line, int nc, int len );
```

complements **insert**. It deletes the indicated line from the file and moves everything beneath it up a notch to fill the gap. It's passed the same information as **export** so that it can save the deleted line for a possible **undo** operation.

The easiest way to manage a file is with an array of **long**s that's indexed by line number and that holds the byte offset from the base of the file to the first character of the indicated line. A delete and insert can manipulate this array. A read goes to the file and simply reads in the line. A write is a bit more complicated, because the lines are probably not all the same length. The best approach is to put the actual text of the overwritten line at the end of the file, rather than trying to insert it into the file itself. You can then overwrite the beginning-of-line pointer in the array with the offset to the

Turn your C/C++ compiler into a powerful scientific programming tool...
C/Math Toolchest
Now you can develop sophisticated scientific and engineering applications using your favorite C or C++ compiler. With a broad range of mathematical functions, the C/Math Toolchest™ gives C as much number crunching power as FORTRAN. More than 135 functions provide support for:
Probability and Statistics: including uniform, normal, binomial, Poisson, and hypergeometric distributions, and least-squares regression.
Complex Arithmetic and Linear Algebra: including a comprehensive set of vector and matrix operations, and solutions to simultaneous linear equations.
Numerical Analysis and Digital Signal Processing: including integration, differentiation, interpolation, root finding, convolution, FFTs, power spectrum analysis, and filter design.
To graphically view the results of your number crunching, we've also included GRAFIX™, a graphical data analysis program. The GRAFIX™ program makes it easy to edit, plot, interpolate, or perform regression analysis of your data.
You may also purchase the C source code for the library and GRAFIX™ program. The C/Math Library Source works with any ANSI standard C or C++ compiler. The C/Math GRAFIX Source works with the C and C++ compilers from Mix®, Borland®, and Microsoft®. Prebuilt libraries for the DOS versions of these compilers are included with the C/Math Toolchest.
Now Only $29.95!
Includes 430 page manual
Order Coupon
Name
Street
City
State Zip
Telephone
❑ Send me free brochures for all Mix products
Paying By: ❑ Money Order ❑ Check ❑ Visa ❑ MC ❑ AmX ❑ Disc.
Card#
Exp. Date
Disk Size ❑ 5 1/4" ❑ 3 1/2"
Qty. Product Price Subtotal
___ C/Math Toolchest $29.95
___ C/Math Library Source $10.00
___ C/Math GRAFIX Source $10.00
Add Shipping ($5 USA, $10 Canada, $20 Foreign)
Texas Residents Add 8.25% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ No Royalties
For technical support, please call:
1-214-783-6001
FAX: 1-214-783-1404
MIX software
C/Math Toolchest and GRAFIX are trademarks of Mix Software, Inc. Mix, Borland, and Microsoft are trademarks of the respective companies.
Circle 50 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

NEW!	Professional IDL (interface generator for complex data; multiple inheritance, ASCII and binary external forms, tools)	$400
	M++ (fast C++ math classes; indexing, matrices & vectors, numerics, memory handling, I/O; specify C++ compiler/Turbo, Zortech, Glock.)	$395
	C++/Views (C++ interface to MS Windows 3.0; over 60 classes; free broswer (no source); no royalties; Zortech C++ only)	$395
NEW!	Embedded DOS (full-features, real-time, multitasking, 3.31-compatible DOS for embedded system and self-bootable installations)	$375
NEW!	Spell Time (spelling checker for incorporation into text products; no royalty; large dictionary; small and fast)	$300
	ZIP Image Processor & Victor Image Library Version 2.0 (brightness, contrast, merge images, TIFF/GIF/PCX/bin, HP ScanJet support)	$290
	TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	TE Editor Developer's Kit for Windows (full screen editor, undo command, multiple windows; inc. TER for application build-in; no royalties)	$220
NEW!	c_pslib (PostScript generation library for C programs; includes complete graphics, font, rotation & paragraph support	$170
	Report Right (report writer and graph generator for Windows 3.0)	$170
	Rogue Wave tools.h++ or math.h++ Class Library (extensive docs)	each $165
	DXF Subroutine Libraries (toolkit for making AutoCAD DXF drawing files; includes manual & sample programs)	$155
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	Minix Operating System (Version 1.5; Unix-like operating system, includes manual; specify 5.25" or 3.5" diskettes)	each $150
	Delorie GCC for MS-DOS (Version 1.39; includes C++, assembler & DOS extender; complete source code and makefiles)	$150
NEW!	CDB for Windows (database toolkit; ISAM, DDL, relational & network, space reuse, concurrent access; fast, manual, no royalty)	$145
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows; with word processing: $195; Windows 3.0: $200)	$130
	C Communications Toolkit by Magna Carta (multi-port & co-processor board support, FAX, interrupt driven, emulations, xfer protocols)	$125
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$120
NEW!	V64 (virtual/extended memory management library, 64Mb address space, fast, direct access to any byte, checkpoint/restart from file)	$120
	WinMem (unlimited global memory handles for Windows 3.0, 4-byte overheadper block rather than 20, virtual memory for 286)	$110
	Network Toolbox/N (C functions for Novell Netware 2.1x, sample programs included)	$110
NEW!	SCM (portable Scheme in C, conforms to IEEE and 3.99 specs, garbage collection off C stack for easy mixing with C)	$100
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Clarkson drivers, SOSS NFS server, Bdale mailer, PCRoute & PCBridge, NDIS, ODI, Beholder, more)	$100
NEW!	VolksWindows (simple graphical interface to DOS file system; copy, launch programs, move files, etc.)	$100
NEW!	Heapman (application memory management for Windows 3.0; 64K bytes of heap space; includes memory browser/debugger)	$100
NEW!	EZTrieve (Novell Btrieve access with data dictionary and data manager; no royalties)	$100
NEW!	CDB for DOS & Unix (database toolkit; ISAM, DDL, relational & network, space reuse, concurrent access; fast, no manual, no royalty)	$95
	PowerSTOR (Version 1.2; extended heap space on extended memory, expanded memory, and/or hard disk)	$95
	MIRACL Multiprecision Integer & Rational Arithmetic C Library (nearly 100 functions; C & C++; from Ireland)	$90
	VMEM (virtual memory manager by Blake McBride, Version 3.6, LRU pager, dynamic swap file, image save/restore)	$80
NEW!	ctase (C database library, sequential & b+-tree random access, buffered block I/O, file-level locking)	$75
NEW!	TreeDraw (PostScript display of labeled hierarchical trees; DOS & Windows screen display included; Moen algorithm)	$75
	COMM-DRV (interrupt-driven serial communication, device drivers & serial port monitors; source for libraries only; complete source $200)	$75
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	FinanC (large collection of financial function including bond, inventory, stock portfolio, & cash flow)	$55
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
	CALC (ASCII algebraic expression evaluator, unlimited parenthesis nesting, symbols, 32 built-in functions, easily extended)	$50
NEW!	Mini IDL (interface generator for complex data; single inheritance, translator & table generator tools, ASCII external form only))	$50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
NEW!	TILE Forth (32-bit Forth-83 in C; lots of extensions including multi-tasking & classes/instances;link in C routines; DOS & Unix)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	C Compiler Pack (5 C compilers; 3 for 8086 (gcc, MicroC, Small C), 2 for 68000 (Sozobon, cc68); gcc ports include library source only)	$50
	GNUish MS-DOS (ports of 17 GNU programs to MS-DOS organized by Thorsten Ohl)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	YATE (Yet Another Text Editor; written in C++, full-screen)	$50
Updated!	CLIPS (rule-based expert system generator, Version 5.0; advanced manuals available at additional cost)	$50
	NIH Class Library & Book (basic C++ classes & *Data Abstraction and Object-Oriented Programming in C++* in softback by Keith Gorlen)	$50
Upgrade!	Editor Pack (fifteen public domain editors; including microEmacs 3.11, Stevie, Elvis, Moke, mg2a, DTE, Jove, & Origami)	$50
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud; domestic distribution only)	$40
	RXC & EGREP Version 2.0 (Regular Expression Compiler and Pattern Matching; finite state machine from regular expression)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	Spell Pack (6 spelling programs, a hyphenator, 2 utility packs and a 60K word list: Ispell, Microsp, Sp, Cspella, Spell, Dawg, Soundex)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	Crunch Pack (30 file compression & expansion programs)	$30
	Make Pack (eight versions of Make including dmake 3.7, GNU Make, Cake, and Gymake and a makefile maker)	$30
NEW!	GNU Ghostscript (Turbo C makefile, MS-DOS executable, Paintjet driver, all fonts)	$25
	BigNum (portable and efficient arbitrary-precision arithmetic package; hardcopy docs; from France)	$25
NEW!	Database Pack (seven databases, simple to complex: BPLUS, AVL, SDB, ID, GDBM, REQUIEM, INGRES89)	$25
	String Pack (lots of string routines; C++ String class, BAWK, word wrap, fuzzy search, Boyer-Moore, Hypertext, etc.)	$25
NEW!	UUPC Pack (UUCP for the PC; UUPC Version 1.11c by Wonderworks and smail/PC Version 2.5 by Stephen C. Trier)	$25
	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language; includes hardcopy docs)	$25
NEW!	Tcl (Tool Command Language; add shell programming programming capability to any command line; elegant command line language)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; BSD Version 2.3.6 with docs)	$25
Updated!	XLISP 2.1 (includes Almy improvements)	$20
	GNU Chess (GNU Version 3.1 & Windows Version 1.01; executable included; requires SDK to build; truly beautiful!)	$20
	GNU RCS (FSF's version of the Revision Control System; like Unix's SCCS only better; keeps track of software development)	$20
	SDBM (fast, disk-based hash table manager for really large hash tables; clone of Unix *ndbm*)	$20
	Unix/386	
NEW!	X-Windows (client & server, shared library, Xlib, executables, XAW etc.; code is diffs on X11R4; TCP or same machine)	$200
	Kyoto Common Lisp (Austin flavor, naturally; Version 1.530; includes GCC and Portable Common Loops (PCL))	$140
	GNU Emacs Version 18.55 (includes complete source & executables)	$125
	Waffle BBS (interface to UUCP & B or C news; internal forums; files section; external processing; full USENET support)	$120
	GNU C Compiler (gcc) Version 1.39 (includes complete source & executables)	$100
	ViewComp Spreadsheet (internal termcap; printer output; hardcopy docs)	$80
	GNU Debugger (gdb) Version 3.5 (includes complete source & executables)	$60
	Gillespie Pascal-to-C (many flavors of Pascal handled; readable & maintainable output)	$25
	Gosling Spreadsheet Pack (variants of the 1982 Gosling spreadsheet including PubliCalc and SC version 6.10; 1 MS-DOS variant)	$25

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3587 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

much more ... ask for catalog

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Circle 51 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
52	American Cybernetics	85
51	Austin Code Works, The	79
44	Autumn Hill Software	68
110	Az-Tech Software, Inc.	94
38	Black Ice Software	58
6	Blaise Computing, Inc.	9
57	Borland International	C4
115	Burton Systems Software	94
*	C Gazette	65
*	C.C. Software	94
36	Catenary Systems	56
120	Cater Software	94
39	Data Techniques, Inc.	58
127	Extra Mile Writing Services	95
112	EZX	94
45	Genus Microprogramming, Inc.	67
*	Gimpel Software	82
7	Greenleaf Software, Inc.	11
35	GUI Computer	56
8	Helpful Programs, Inc.	91
118	Impulse Engineering	94
129	InSync Software	95
113	Interstitial Software Systems	94
27	Kadak Products, LTD	44
31	Kandu, Inc.	53
10	Knowledge Dynamics Corp.	13
49	Lahey Computer Systems, Inc.	75
56	Meridian Software Systems	C3
5	MetaWare Incorporated	7
124	MicroQuill	95
*	Microsoft Corporation	3
*	Microsoft Corporation	19
55	Microsoft Press	90
128	Microtools	95
*	MicroWay	69
50	Mix Software	78
15	Modular Software Systems	16
18	Nanosoft	25
4	Nu-Mega Technologies	4
3	Olivadoti Associates, Inc.	2
111	Opt-Tech	94
12	Opus Software	14
33	Pathfinder Associates	54
47	PC Tech, Inc.	66
26	PC Techniques	42
17	Periscope Company, The	23
30	Petrasoft	53
*	PJ Bulletin Board	48
*	PJ Mail Order Store	81
19	Programmer's Corner, The	25
*	Programmer's Journal	57
*	Programmer's Journal	95
46	ProGraphx Toolbox	66
125	Quaid Software Limited	95
48	Qualitas, Inc.	73
24	Quantasm Corporation	36
1	Quarterdeck Office Systems	C2,1
*	Que Corporation	83
22	RSS Engineering	34
117	Ryle Design	94
14	Sequiter Software, Inc.	15
*	Set Laboratories, Inc.	18
34	SLR Systems	55
29	Software Annex, The	48
9	Software Blacksmith	91
119	Software Mart	94
54	Software Success	86
121	Software Toolz, Inc.	95
42	Softway, Inc.	62
123	Strategic Software Designs Inc.	95
40	Sub Systems	59
126	Sutrasoft	95
41	TEGL Systems Corp.	59
23	Tool Makers, The	35
13	Tools & Techniques, Inc.	14
114	TSR Systems Limited	94
25	TurboPower Software	39
122	Upper Deck Systems	95
21	Utilis, Inc.	29
*	V Communications	47
43	Weitek Corp.	63
116	Wenham Software Co.	94
32	Western Wares	54
11	Winsoft	13
53	Willies' Computer Software	86

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

EDITORIAL INDEX

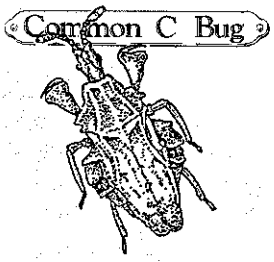
Reader Service	Vendor Name	Page Number
177	Blaise Computing Inc.	93
213	Connectix	44
167	Crescent Software, Inc.	92
214	Cyrix Corporation	75
175	Digitalk, Inc.	93
212	Elite Microelectronics	18
170	Emerging Tech Consultants	92
219	Ergo Computing	36
217	Integrated Information Tech	75
215	Intel Corporation	75
220	Intel Corporation	36
169	Lahey Computer Systems	92
221	MetaWare	36
168	Microsoft Corporation	92
222	MicroWay, Inc.	36
172	MicroWay, Inc.	92
211	Motorola	18
223	OTG Systems, Inc.	36
166	OTG Systems, Inc.	92
224	Phar Lap Software	36
174	Quantum Software Systems	93
227	Rational Systems	36
225	Silicon Valley Software	36
171	SLR Systems	92
182	Softran Corporation	93
176	Software Science, Inc.	93
218	Specialty Development Corp.	75
173	Spirit of Performance, Inc.	93
178	TurboPower Software	93
179	Vleermuis Software Research	93
226	Watcom	36
180	Watcom	93
216	Weitek Corporation	75
181	Whitney Software, Inc.	93

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

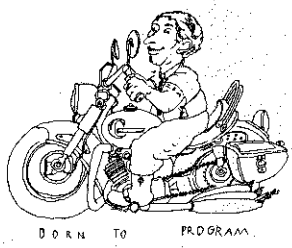
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$14.95 each — includes U.S. shipping



COMMON C BUG



BORN TO PROGRAM

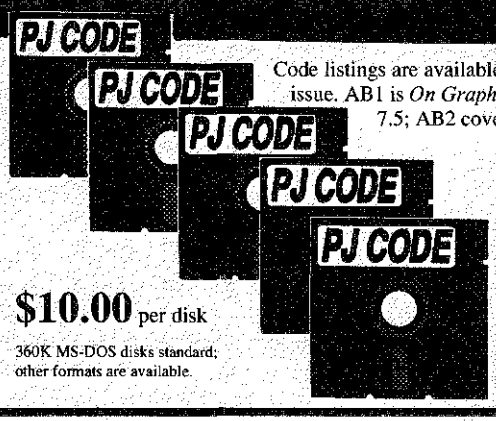
PARALLEL PORTS

RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00 per disk

360K MS-DOS disks standard; other formats are available.

Code listings are available from volume 5.5 to the current issue. AB1 is *On Graphics* series code from issue 5.1 to 7.5; AB2 covers issue 7.6 to the current issue.

Diskettes are available for the following code:

AB1	6.4	7.5	8.6
AB2	6.5	7.6	9.1
5.5	6.6	8.1	9.2
5.6	7.1	8.2	9.3
6.1	7.2	8.3	9.4
6.2	7.3	8.4	9.5
6.3	7.4	8.5	

PJ ISSUE SPECIAL!

SAVE $10.00 ON ABRASH BOOK & DISK

Power Graphics Programming

ABRASH BOOK AND CODE DISK

For a limited time, you can save over 25% when you order both the *Power Graphics Programming* book and the companion code disk during the *PJ* 9.5 Issue Special. Lowest price ever! Order your book and disk today for only $29.90 and save $10.00.

(Offer expires October 31, 1991. All orders must be prepaid.)

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; Graphics programming in windows; Mousing around: Messages in PM; Accessing high memory; ASM: Advanced segmentation; Ellipses on the VGA; Up OOP alley in Pascal.

Vol. 8.4: PROGRAMMING IN THE PROTECTED MODE: Faster VGA ellipses and more; Running unprotected virtual 8086 mode; Converting to 386 protected mode; OS/2 2.0 memory model for the 386; Fortran for the 386; Using C++ features in C; Different times compensating for system clock idiosyncracies; ASM: Errors and stream I/O; Cobol-85 nested; Deliberately breaking the rules.

Vol. 8.5: OBJECT_ORIENTED PROGRAMMING: More colors on the VGA; A virtual memory system in C/C++; Pitfalls in Turbo Pascal OOP; Writing applications that travel; Late binding; Tools for 386 DOS; Windowing text with X; Cache memories; Breaking the rules again; SPARCs plot world takeover.

Vol. 8.6: THE DATABASE CHALLENGE: VGA color paging; The database dilemma; Networking databases; B-tree indexing; Virtual memory paging; Writing an ANSI C preprocessor; Peak performance: On to the 80486; Calendar conversions; Software components; Optimization strategies; Pink noise; Oh say can you C?

Vol. 9.1: GRAPHICS AND WINDOWING: VGA color cycling; Digital typography; Windowing for two; GUI madness; Text buffers in windows; What's on the menu; 386 DOS meets ANSI C; Optimania; Real-mode virtual memory; Enhanced keyboards; An open look at Motif; Negotiating tricks and traps; For a few good books; Goliath beats David—As usual.

Vol. 9.2: LANGUAGES: Windows 3 input hooks; C compilers for Windows; Fortran 90 is coming!; Pragma interface; String handling; Porting to the Mac; Disassembler roundup; Exploring XMS; Using EMS for fast file I/O; Around the bend again; Pink noise; Language lore; Effective marketing presentations; COBOL revisited; Language parsers and Fortran tools; Changes in the fast lane.

Vol. 9.3: HARDWARE FOR DEVELOPERS: Pixel panning; Hardware-assisted debuggers; Object-oriented SCSI programming; Numeric exception handling; Virtual realities; BASIC libraries; BIOS compression; Hashing functions; Computer stores and popcorn kernels; Programmer productivity; Language translation and 386 debugging; Hardware help; Hardware decisions.

Vol. 9.4: NETWORKING AND COMMUNICATIONS: A coat of many colors; Communicating over NetBIOS; Spying on NetBIOS; Window viewports; Loop unrolling: A RISCy business; Data structure imperatives; Text filters in BASIC; Pink noise; Don't be wrong about your rights; Network madness; Network debugging and comm libraries; Benchmark SPECulations.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

new (rather than to the original) text. An occasional "garbage collection" pass can go through the array and create a second file that contains only the active lines (the ones with pointers to them in the array). It then replaces the current file with the new one and adjusts all the pointers to hold the correct offsets. The Unix vi editor does this every time a **write** operation is performed.

You can also store a copy of your array of **longs** in the file. This lets you recover a file if a power failure (or some other catastrophic error) occurs in the middle of an editing session. Also note that the "file" need not be a real file—it can be on a RAM disk or in extended or expanded memory. This can save considerable time when you do reads, writes, or refreshes.

BYPASSING THE EDITOR

The rest of WINDOW.H contains function prototypes for those window functions that aren't used by the editor. Listing 3 demonstrates how these functions can be used for a simple editor/window application.

The call to

```
win *w_construct (int win_height,
                  int win_width );
```

creates the window structure, initializing the underlying buffer to the indicated height and width. The window is not displayed, however. Various window attributes are modified as necessary.

```
void w_viewsize( win *w,
                 int view_height, int view_width );
```

modifies the viewport size from its default (the entire window) to the indicated height and width.

```
void w_viewcolor ( win *w,
                   int foreground, int background );
```

modifies the foreground and background colors from the defaults (light gray background, black foreground). The possible colors are defined in Borland's CONIO.H file. Next,

```
void w_move( win *w, int row, int col );
```

moves the viewport's position on the screen from the default (banged up against the upper-left corner) to the indicated row and column. The upper-left corner is (0,0). **w_move** can be called at any time to move the window around on the screen. The

```
void w_file( win *w, ... );
```

call attaches file-manipulation subroutines to the window so that an underlying file can be accessed. If you don't call this routine, then the window functions assume that no underlying file exists.

Finally,

```
void w_open( win *w );
```

actually displays the window on the screen.

When the function is finished with the window, a call to

```
void w_destroy( win *w );
```

removes it from the screen, restoring the underlying text and freeing all memory associated with the window. The text buffer is also flushed to disk at this time.

The remaining high-level functions include

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

PC-lint presents C Bug # 747

```
#if PROTOTYPES
    double sqrt (double);
#else
    double sqrt ();
#endif

main ()
    { printf ( "sqrt(2) = %g\n" , sqrt(2) ); }
```

It looks like the programmer is being careful with his prototyping, allowing his compiler to check his calls if it supports prototypes and allowing for the occasional compiler that doesn't. But, what's wrong? When does this fail? Call if you need a hint.

PC-lint will catch this and many other C bugs. Unlike your compiler, PC-lint **looks across all modules** of your application for bugs and inconsistencies.

More than **270 error messages.** More than **90 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc. **Easy to Use** – If you know C you already know how to use PC-lint. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI prototypes for your K&R functions.

Attn: Power users with huge programs.

PC-lint 386 uses DOS Extender Technology to access the full storage and flat model speed of your 386.

PC-lint 386 – $239
PC-lint DOS or OS/2 – $139

Mainframe & Mini Programmers

FlexeLint in shrouded source form, is available for Unix, OS-9, VAX/VMS, QNX, IBM VM/MVS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.

Gimpel Software

3207 Hogarth Lane, Collegeville, PA 19426

CALL TODAY (215) 584-4261 Or FAX (215) 584-4266

30 Day Money-back Guarantee.

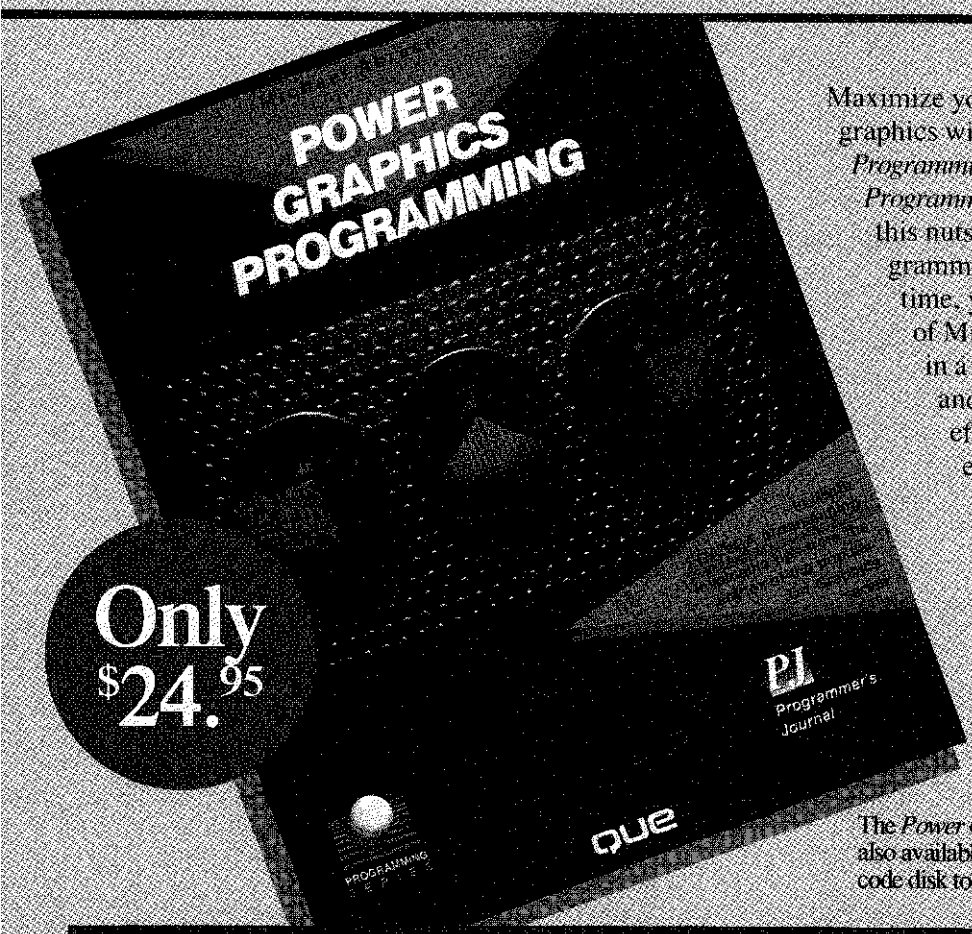
PA add 6% sales tax.

PC-lint and FlexeLint are trademarks of Gimpel Software

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Results!

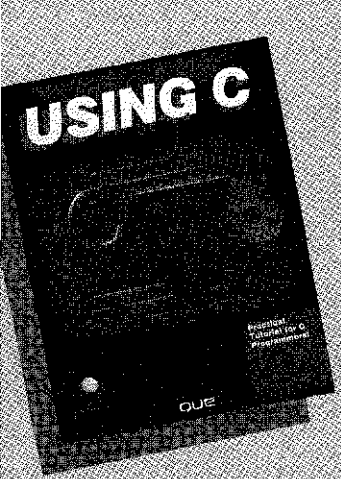


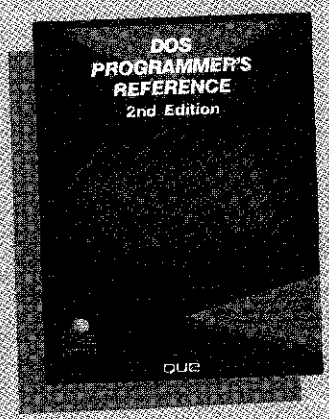
Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming*! Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming*!

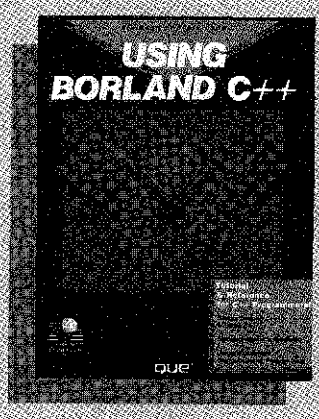
Power Graphics Programming is only available from *Programmer's Journal*. Call 800-234-0386.

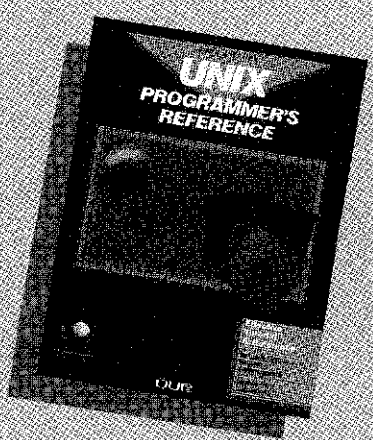
The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!

Look for these other practical Programming books from Que









Available in better bookstores and computer stores everywhere.
Or call 800-428-5331, ext. PJ01.

que®

PROVIDED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
void w_hide( win *w );
```

which hides the window (makes it vanish from the screen) without destroying it. You can write to a hidden window, or you can move it. Redisplay it with another **w_open** call. The

```
void w_move_rel( win *w,
        int delta_row,
        int delta_col );
```

moves the viewport relative to the current position on the screen rather than to an absolute position. A negative **delta_row** goes left; a negative **delta_col** goes up.

The final function prototype is for

```
void wputc( win *w, int c );
```

There is no **wputc** in the current article, but the prototype allows you to add one. The most basic **wputc** simply passes characters to the editing engine as they come in from the keyboard.

All these functions are declared in WINDOW.C (Listing 4).

CONCLUSION

So that's all for now. This powerful windowing system has the potential to let you manipulate huge text buffers that are stored in files through a tiny viewport on the screen. Next time, I'll provide windowing support for the editor. The main deficiency in the current implementation is error checking, which could be improved. You could also expand the window concept considerably, implementing such things as boxed windows and subwindows. (When you move a window, all the subwindows move with it. This is useful if you're using small windows to implement fields in a larger screen.) Even so, I hope you find the functions provided here helpful as they stand. PJ

LISTING 1

```
/* b_last.c-- Find Last Character on Line */
#include <tools/textbuf.h>

b_last( textbuf *b )
{
    bufptr startp = b->buf + (b->ncols * b->row);
    bufptr endp   = startp + (b->ncols);
    while( --endp > startp && isspace(*endp) )
        ;
    return( endp - startp );
}
```

LISTING 2

```
/* window.h-- High-Level Window Definitions */
#ifndef __WINDOW_H
#define __WINDOW_H

#include <tools/textbuf.h>      /* textbuf definition.  */
#include <tools/viewport.h>     /* engine definition.   */

#define ETYP struct win
#include <tools/editor.h>       /* Customize editor.h prototypes  */
                                /* for use in a window.           */
#define W_LINEMAX E_LINEMAX     /* This can be eliminated if      */
                                /* not using an editor layer.     */
typedef struct win
{
    editor e;           /* Variables for engine maintenance       */
                        /* This would be the base class in a C++  */
                        /* implementation. THIS MUST BE AT        */
                        /* THE TOP OF THE STRUCTURE BECAUSE IT'S  */
                        /* USED POLYMORPHICALLY BY THE EDITOR.    */
                        /* This can be eliminated if you're not   */
                        /* using an editor layer.                 */
    textbuf  b;         /* Text buffer and viewport.              */
    viewport v;         /* (instance variables, not base classes) */

    unsigned row_off;  /* Offset to upper-left corner of textbuf   */
    unsigned col_off;  /*        to upper-left corner of viewport  */
    unsigned file_off; /* Offset from top of file to top of buffer.*/
                       /* Initially zero, incremented on a success-*/
                       /* ful import when text scrolls up. Decre-  */
                       /* mented on import when text scrolls down. */
                       /* I'm assuming that text is positioned     */
                       /* initially at the top of the file, so     */
                       /* it's an error if you try to scroll text  */
                       /* down and file_off==0.                    */
    int bottom;        /* Line number of bottom line in file. -1   */
                       /* if the file is empty.                    */

    unsigned update_row_off;  /* Position when screen updating is  */
    unsigned update_col_off;  /* disabled                          */
    unsigned update_file_off;
    unsigned update_row;
    unsigned update_col;

    unsigned update : 1; /* Viewport updating is enabled.          */
    unsigned dirty  : 1; /* File modified when updating disabled.  */
    unsigned error  : 1; /* Error has occurred at some point.      */

    unsigned engine_active : 1; /* Editing engine initialized      */

    char *(*import)(struct win *this,int lnum );
    int   (*insert)(struct win *this,int lnum );
    void  (*export)(struct win *this,int lnum,char *line,int nc,int len);
    void  (*remove)(struct win *this,int lnum,char *line,int nc,int len);
}
win;

win *w_construct ( int win_height, int win_width );
void w_destroy   ( win *w );
void w_open      ( win *w );
void w_hide      ( win *w );
void w_viewsize  ( win *w, int view_height, int view_width );
void w_viewcolor ( win *w, int foreground, int background );
void w_move_rel  ( win *w, int delta_row, int delta_col );
void w_move      ( win *w, int row, int col );
void wputc       ( win *w, int c );
void w_file      ( win *w,
    char *(*import)( win *this,int lnum ),
    int   (*insert)( win *this,int lnum ),
    void  (*export)( win *this,int lnum, char* line,int nc,int len),
    void  (*remove)( win *this,int lnum, char* line,int nc,int len)
                );
#endif /* __WINDOW_H */
```

LISTING 3

```
/* wintest.c-- Test main() for Window-Based Editor */
#include <stdio.h>
#include <conio.h>      /* Borland video defines (RED, CYAN, etc.) */
#include <tools/window.h>

/* Hooks for the file manager. Fill these in if you want an       */
/* underlying file. If not, remove the w_file() call in main().  */

#pragma argsused
static void sigexport( win *this, int line_number, char *line,
                                                int ncols, int len)
{
}

#pragma argsused
static char *sigimport( win *this, int line_number )
{
    static char line[128];
      return line;
}

#pragma argsused
static void sigremove(win *this,int line_number,char *line,
                                                int ncols, int len)
{
}

#pragma argsused
static int  siginsert( win *this, int line_number )
{
}
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
#define WINSIZE  20     /* 40x40 text buffer */
#define VIEWSIZE 10     /* 10x10 viewport    */

void main()
{
    static int c;
    static win *w;

    w = w_construct(   WINSIZE,  WINSIZE        );
    w_viewsize      (w, VIEWSIZE, VIEWSIZE      );
    w_viewcolor     (w, RED, CYAN               );
    w_move          (w, 1  , 1                  );
    w_file          (w, sigimport, siginsert, sigexport, sigremove);
    w_open          (w );

    while( (c = getch()) != 'q' )
           wputc( w, c );
    w_destroy( w );
}
```

LISTING 4

```
/* window.c-- High-level Window Functions */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <tools/window.h>

void wputc( win *w, int c );    /* Test function in wputc.c */

#include <conio.h>              /* Debugging stuff          */
#include <dos.h>

static int Instance_count = 0;

win *w_construct( int win_height, int win_width )
{
    win *w;
    if( !(w = malloc(sizeof(win)) ) )
        return NULL;

    if( ++Instance_count == 1 ) /* Creating first window        */
        v_startup( C4350 );     /* Start up video system.       */

    w->row_off  = 0;    /* Viewport in upper left corner of     */
    w->col_off  = 0;    /* the buffer.                          */
    w->file_off = 0;    /* No lines above window in file        */
    w->bottom   = -1;   /* No lines read yet                    */
    w->import   = NULL; /* No overflow file by default.         */
    w->export   = NULL;
    w->remove   = NULL;
    w->insert   = NULL;
    w->engine_active    = 0;    /* engine not initialized       */
    w->update           = 0;    /* updating is disabled         */
    w->update_row_off   = 0;    /* when updating is enabled,    */
    w->update_col_off   = 0;    /* set cursor to top left of    */
    w->update_file_off  = 0;    /* viewport, buffer, and file.  */
    w->update_row       = 0;
    w->update_col       = 0;
    w->dirty            = 0;

    /* Create text buffer and viewport */
    b_construct  ( &w->b, win_height, win_width );
    v_construct  ( &w->v );
    v_saveok     ( &w->v, 1 );
    v_clearok    ( &w->v, 1 );
    v_size       ( &w->v, win_height, win_width );
```

continued

Editor Headaches?
SWITCH to the COMFORT and FREEDOM of
multi edit
At a
DO IT NOW!
Don't put it off! This offer is only good through 12/31/91.
TEL: (602) 968 1945
FAX: (602) 966 1654
1930 W. University Dr. •Suite 112
Tempe, AZ 85281
Still not sure...?
Get our FREE, FULLY FUNCTIONAL DEMO!
LIST PRICES $129. Standard $199. Professional
C.O.D.
AMERICAN CYBERNETICS
Multi-Edit is a trademark of American Cybernetics, Inc. Other trademarks are the property of their respective holders.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROFESSIONAL SERIAL COMMUNICATION MADE EASY

DON'T BE FOOLED BY THE PRICE

COMM-DRV is a professional serial communication library that comprises a complete serial I/O library, a true MSDOS RS232 device driver, an RS232 TSR, a serial port monitor, and a dumb terminal emulator. It is second to none!!!

VERY EASY TO USE !

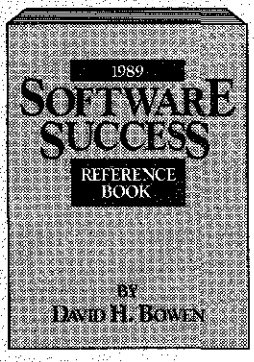
- 25+ serial ports sharing one or more IRQs. Remap COM ports to any address.
- Supports speeds up to 115.2 kilobaud.
- Hardware/Software Handshake - RTS/CTS DSR/DTR XON/XOFF
- Installable as a device driver or as a TSR (can be de-installed from RAM).
- Can be linked into application or used via TSR or device driver.
- Supports adjustable communication buffer sizes on every port.
- Runs under Desqview, Omniview, VPIX, Windows 3.0, and most multitaskers.
- Interface to DOS via INT21, INT14, C's open(), read(), write(), close(), ioctl().
- Selectively interrupt user routine on reception, transmission, & status change.
- Supports most multiport cards including Digiboards & Intel Hub card.
- Coexistence of different multiport, singleport, & smart cards.
- Supports the FOSSIL interface used on most bulletin boards (e.g. OPUS)
- Supports all file transfer protocols.
- 8250, 16450, 16550 (including FIFO buffers), etc. support.
- Compatible with all languages (C, QBASIC, PASCAL, COBOL, etc.)
- Customization provided upon request.

Driver, Library, Sources $89.95
Driver, Library, Sources, & 4 PORT SERIAL CARD $199.95
Windows 3.0 Driver For Multiport Cards $99.95

TEL.(713)498-4832 FAX (713)-568-3334
BBS (713)-568-6401
1-800-966-4832
VISA/MC/AMEX/PO/COD/CHECK 24 Hours/Day 7 Days/Week
Willies' Computer Software Co.
2470 S. Dairy Ashford, Suite 188 Houston, Tx. 77077

Circle 53 on Reader Service Card

THE ANSWER BOOK FOR YOUR SOFTWARE MARKETING PROBLEMS!



SOFTWARE SUCCESS REFERENCE BOOK 1989 by David H. Bowen

I bet at least once today, you've already been faced with a tough decision in some area of your business. Was it Promotion? Lead Generation? Sales? Pricing? A Legal or Management issue? Don't you wish you could pull a book off your shelf and, within seconds have a solution to your problem? Does such a book exist? YOU BET IT DOES!

It's THE SOFTWARE SUCCESS REFERENCE BOOK, a 262-page guide, organized by topic to provide you with FAST HELP in solving your TOUGH PROBLEMS. Compiled from a full year of *SOFTWARE SUCCESS* — the "whole business" newsletter for software CEOs — this indispensable answer book will save you hours of needless worry over making the right decisions for your software company.

Send your check for $50; or call or FAX with your credit card payment (VISA/MC/AEX).
BY PREPAYING, YOU'LL RECEIVE A FREE 3-MONTH SUBSCRIPTION to *SOFTWARE SUCCESS* — a $57 value!

SOFTWARE SUCCESS
P.O. Box 9006
San Jose, CA 95157
Ph: (408) 446-2504
Fax: (408) 255-1098

Circle 54 on Reader Service Card


```
    v_foreground ( &w->v, BLACK        );
    v_background ( &w->v, LIGHTGRAY    );
    return w;
}
void w_destroy( win *w )        /* Destroy the window */
{
    v_destroy  (&w->v);         /* Destroy the viewport          */
    b_destruct (&w->b);         /* Destroy the text buffer       */
    free( w );                  /* Free the space for the window */
    e_close( w );               /* Flush buffer to underlying file */
    engine_close( w );          /* Shut down the editor.         */
    if( --Instance_count == 0 ) /* Destroyed the last window.    */
        v_shutdown  (   );      /* Shut down the video system.   */
}
void w_open( win *w )
{
    v_open  ( &w->v );          /* Display the viewport          */
    e_update( w, 1  );          /* Enable writes to the window   */
                                /* and position cursor at (0,0)  */
    if( !w->engine_active ){    /* Edit engine is uninitialized. */
        e_open( w );            /* Initialize buffer             */
        engine_open( w );       /* Initialize edit engine.       */
        w->engine_active = 1;   /* Mark it as initialized.       */
    }
}
void w_hide( win *w )
{
    e_update( w, 0 );           /* Disable viewport updating.    */
    v_deactivate( &w->v, 1 );
}
void w_viewsize( win *w, int view_height, int view_width )
{
    v_size( &w->v, view_height, view_width );
}
void w_viewcolor( win *w, int foreground, int background )
{
    v_foreground ( &w->v, foreground  );
    v_background ( &w->v, background  );
}
void w_move_rel( win *w, int delta_row, int delta_col )
{
    /* Move delta_row rows down and delta_col columns to the right. I'm
     * assuming that the window's hidden if updating is off. This might
     * not be the case, though. If this assumption causes problems, add
     * a "hidden" attribute to either the viewport structure
     * (preferable) or the win structure (less preferable, but it will
     * work). If you do the former, provide a v_visible() that
     * evaluates true if the viewport is visible. */

    if( !w->update )
        v_move_rel( &w->v, -delta_row, -delta_col );
    if( w->update )
    {
        w_hide( w );
        v_move_rel( &w->v, -delta_row, -delta_col );
        w_open( w );
    }
}
void w_move( win *w, int row, int col )
{
    /* Move delta_row rows down and delta_col columns to right */
    if( !w->update )
        v_move( &w->v, row, col );
    else {
        w_hide( w );
        v_move( &w->v, row, col );
        w_open( w );
    }
}
void w_file( win *w,
        char *(*import)( win *this, int lnum ),
        int  (*insert)( win *this, int lnum ),
        void (*export)( win *this,int lnum,char *line,int nc,int len),
        void (*remove)( win *this,int lnum,char *line,int nc,int len)
      )
{
    /* Attach file-level support to the window functions. */
    w->import   = import;
    w->insert   = insert;
    w->export   = export;
    w->remove   = remove;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pursuing systems programming

By M. Steven Baker

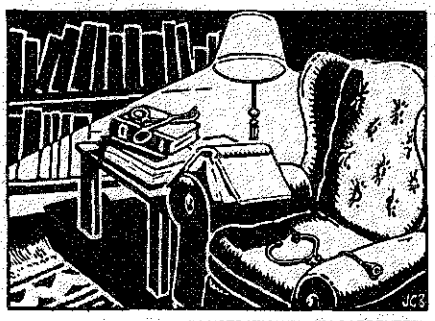


ILLUSTRATION BY JIM CARPENTER

Challenging problems have always attracted me, so over the years I've been drawn to systems programming—first under CP/M-80 and then in the DOS environment. These days a PC programmer might need fluency in the Mac world, the 386 Unix market, and on workstations (aka Sun). Considering the vast array of add-on hardware and ever-changing system software, there's still a great demand for systems programming. This time around I'll review some newer books that cover PC systems programming, and I'll cross over onto the other turf—that of the VAX world and the big IBM mainframes.

THE NEW DOS ARRIVES

In the past, the PC community has depended on IBM to provide technical reference manuals on PC hardware and DOS system software. But with Windows and DOS 5, IBM is no longer the only source of information for DOS software. Fresh from the printer, the ***Microsoft MS-DOS Programmer's Reference,*** from Microsoft Press, is a technical reference for DOS 5. Written by Microsoft staff, this book covers superseded features from earlier DOS versions, but its real value is its coverage of the new DOS.

The book starts with an introduction and overview of DOS, including chapters on the DOS file system, character input and output, program management (resources, memory, TSRs, etc.), and national language support. The bulk of the book is the last three chapters (appendices in earlier IBM DOS manuals), which discuss DOS interrupts (other than INT 21h), the INT 21h functions, and device drivers. Appendices cover code pages, extended key codes, and error values.

This official DOS 5 reference manual makes a modest break from the past by documenting some previously undocumented DOS features. For example, you'll find brief information on the interrupts that TSR programs depend on. These include INT 28h (DOS idle handler), INT 21h function 32h (**get InDos flag**), and DOS's **ErrorMode** flag. The book covers several of the DOS multiplex interrupt handlers (INT 2fh functions) as well as the new task-switching and memory-allocation features, system calls, and other new functions. For example, DOS 5 has a new **get version** call (INT 21h called with AX = 3306h) that returns the actual DOS version, because the DOS 5 SETVER utility allows you to fool a program. This call also returns a revision number and indicates whether DOS is in the high memory area or is running from ROM. If you want to take full advantage of DOS 5, this reference is mandatory.

Abacus has a valuable programmer's resource in ***PC System Programming,*** by Michael Tischer (1990). An in-depth reference for DOS programmers, this rather massive book is divided into five parts: an overview of PC components; DOS; BIOS; PC hardware; and the interaction among keyboard, BIOS, and DOS. Large appendices provide detailed reference to DOS and BIOS functions. Some undocumented DOS features are well illustrated. For programmers, this book's advantage is its combination of DOS, BIOS, and hardware coverage in a single volume, profusely illustrated with useful example code. The discussion of DOS is solid, and the chapters covering the ROM BIOS and PC hardware are especially strong.

The book's two companion disks include all the sample source programs in BASIC, C, Pascal, and assembly language. If you're looking for a routine to explore floppy diskettes directly or to reprogram the AT real-time clock, then this book is for you.

YOUR MOUSE, OR MINE?

Another new arrival is Microsoft Press's second edition of their ***Microsoft Mouse Programmer's Reference.*** This edition—an overview and technical reference manual for programmers on the Microsoft mouse—adds coverage of the new functions added in version 8.0 of the Microsoft mouse driver (some calls were added with the 7.05 driver). These new mouse functions (functions 37–52) add support for acceleration curves, virtual coordinates, changing video modes, ballpoint mice, initialization, and hardware reset. Conveniently, mouse function 50 returns a 16-bit flag with a bit set for each advanced function that the driver supports (starting with function 37). The book provides function call examples in interpreted BASIC, QuickBASIC, C, and assembly language, with some interface examples for Fortran and Turbo Pascal. Also discussed are the Microsoft mouse menu language and the MOUSE.LIB EGA interface. Two disks are included, with sample programs written in BASIC, C, Fortran, Pascal, and assembly language.

With IBM planning a fall marketing blitz, OS/2 is receiving renewed attention. One of the better books on OS/2 graphic interface programming is ***OS/2 Presentation Manager Programming Primer,*** by Asael Dror and Robert Lafore (Osborne McGraw-Hill, 1990). This introduction to PM programming is divided into four parts: an overview, the user interface, graphics and text, and advanced topics. The section on the user interface has a competent discussion of windows, messages, controls, resources, menus, and dialog boxes, but the strongest chapters are those on graphics and the OS/2 GDI. The authors also illustrate the use of the clipboard, multitasking, and foreign language support. The book is heavily laced with example OS/2 source code

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in C (which is available on disk from the authors). This is the best book I've found on the subject of OS/2 PM graphics programming.

UNIX AND BEYOND

An interesting older (1987) book from Prentice-Hall is ***Operating Systems: Design and Implementation,*** written by Andrew Tanenbaum. Although most books on operating systems are long on theory, this book tries to balance theory and practice. The author examines the fundamental concepts—including process management, memory management, input/output, and the file system—in reasonable detail. But the focus is on MINIX, a Unix-compatible operating system. MINIX supports the same system calls as the older Version 7 of Unix; a complete, heavily commented source code listing of MINIX is provided for study. You'll find both detailed discussion and source code examples that illustrate process management, interprocess communication, message passing, scheduling algorithms, semaphores, deadlocks, memory management, paging algorithms, file system design, input/output, device drivers, security and protection mechanisms, and more.

In addition to Tanenbaum's book, Prentice-Hall sells versions of MINIX on disk with manuals for the IBM PC, Mac, Atari ST, Amiga, and SPARC. Along with the kernel, MINIX includes a Unix-compatible shell, utility programs, and a C compiler. These disk versions include the complete source code for the operating system as well as all the utilities except the C compiler and assembler.

Most operating systems, particularly Unix, are moving to comply with Posix—an ANSI standard under development that defines an operating system interface for C programs. ***The Posix.1 Standard: A Programmer's Guide,*** by Fred Zlotnick (Benjamin Cummings, 1991), explains the syntax and semantics of the 200+ C functions and macros covered in the Posix.1 standard and the data structures that support them. These data structures include process and system attributes, process creation and synchronization, files and directories, I/O, signals, controlling terminals, and data interchange formats. Offering detailed discussion, C function prototypes, and example code, the book is well organized and easy to read.

Wiley's ***Berkeley UNIX: A Simple and Comprehensive Guide,*** by James Wilson (1991), provides a compact introduction to the user and programming interfaces for Berkeley Unix. The book is divided into three sections: Unix commands and user interface, shell programming, and the C programming interface. Both the Bourne and the C shell are modestly covered. I found the chapters on creating multiple processes (forks), pipes, signals (interrupts), and interprocess communications (sockets) to be most valuable. Each chapter includes short examples to illustrate the concepts discussed. If you need a reader-friendly introduction to forks, signals, and sockets, I highly recommend Wilson's *Guide.*

IT'S A MAINFRAME WORLD?

If I'm at all typical, many a PC programmer still has one foot in the minicomputer/mainframe world, whether they want to or not. If you also occasionally work on big IBM iron (IBM 360/370/390, etc.), the third edition of ***System 370/390 JCL,*** by Gary Brown (Wiley, 1991), is an excellent resource. IBM mainframe operating systems are a world unto themselves, and the official IBM manuals aren't very useful. This book provides comprehensive and easy-to-read coverage of IBM's Job Control Language (JCL) on the MVS operating system. After a brief introduction, various JCL statements are covered in separate chapters, illustrated with short examples of usable JCL code. Brown writes with clarity and passion as he explains JCL, but he doesn't try to make you like it.

Beyond JCL, the book also examines several IBM utility programs, VSAM data sets, the linkage editor, ISPF/PDF (the IBM text editor), TSO, and TSO/REXX. The author also describes how to use the Fortran and the VS COBOL II compilers. This new edition covers the latest changes and extensions to JCL and the 390 architecture. Bound in the back is a handy quick-reference guide to JCL statements and syntax. If datasets and DD statements are getting you down, this book might provide a potent elixir. If you work on IBM mainframes, consider it an indispensable resource.

JUST THE VAX

I'd also like to mention two books for the VAX crowd. ***Mastering VMS,*** by David Bynon (Professional Press, 1990), is a solid reference on the VAX VMS operating system. For those unfamiliar with the VAX, both CP/M-80 and, consequently, many DOS utility programs and batch facilities were modeled on VMS. The first section of this book—an overview of the VAX, VMS, DECnet, and VAX clusters—is followed by chapters on VMS utilities and commands, command procedures, DECwindows, operational management, security, and performance. The VMS shell, which is VAX's powerful Digital Command Language (DCL) interpreter, is discussed in detail. The author illustrates the use of the VMS text editors (EDT, EVE, and TPU), the online HELP utility, and the MAIL and PHONE programs.

The chapters on management, security, and performance are aimed at system programmers who are responsible for maintaining a VAX system, but the information is also valuable for other programmers. The comprehensive coverage of DCL and the reference material on EDT and EVE commands make this a valuable resource for occasional VAX programmers.

Those who can find beauty in details will appreciate Digital Press's 1991 ***VAX/VMS Internals and Data Structures,*** written by Ruth Goldenburg and Lawrence Kenah. This massive tome was written by the folks from DEC who helped develop VMS. The latest edition covers the VAX/VMS version 5.2 operating system. Chapters discuss subjects such as control mechanisms, synchronization, scheduling and time support, memory management, input/output, processes, and system start-up and shutdown. As they explain VMS, the authors emphasize the data structures that are manipulated by the kernel along with the control flow of major system routines. Be forewarned: this is not light and easy bedtime fare—the index alone is almost a hundred pages long. But if you need details, this hefty volume supplies them in spades. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Thinking about project management

By William H. Roetzheim

Software can be developed either in a data processing department (or other organizational unit) that is set up in a functional organizational structure, or by using a project approach. For many organizations, especially those whose "software development" is primarily concerned with maintenance and enhancement, the additional management costs and disruptions caused by project management are neither necessary nor desirable. The decision to implement formal project management should be weighed carefully. The following questions can be used to evaluate the desirability of formal project management for your organization:

• Does the job require the integration of many components into a functioning, operational whole? The interaction between these components is best handled by a single individual who has ultimate responsibility for the project. Modular software development, by its nature, is a classic example of this type of complex interaction. Most computer hardware and systems projects have similar requirements.

• Is the job large or technically complex? Such projects usually require and deserve the close attention that management gives to the question of formal project management.

• Does management feel strongly that a single individual should be available to serve as a focal point for information and responsibility? Are tight budgetary or schedule controls necessary or desirable?

• Is the project being developed in an environment of changing requirements? If so, a project management approach will permit faster responses to changing needs and result in greater customer satisfaction.

ILLUSTRATION BY JIM CARPENTER

• Does the task require that a diverse group of professionals be brought together to work toward a common goal? Many software development projects require software professionals to work hand-in-hand with application specialists and end users. Such interactions among different groups are usually best handled in a project management environment.

Although the decision to use formal project management should not be automatic, most software development projects are ideal candidates for this approach. Project management centralizes overall responsibility for the project in one person. The project manager is responsible for all aspects of the project, including budgeting, cost control, scheduling, resource allocation, technical quality, and customer relations. The result is a significantly higher probability of success for the development project.

THE PROJECT IS UP TO YOU

A successful implementation of the project management concept depends on the clear-cut assignment of responsibility for all aspects of the project to one individual—you. The overall outcome of the project rests solely on your shoulders. You are responsible for budgeting as well as cost control. You must schedule tasks, allocate resources (primarily people), and track project expenditures. You are responsible for the technical quality of the work and for managing client/customer and company relations throughout the project. If the project fails, all fingers will point at you.

The single most important area of responsibility is the management of customer relations. Budgeting and cost control are important, but good customer relations can allow you to obtain additional funding to compensate for a shortfall. A happy customer tends to overlook your mistakes and concentrate on your successes, but an unhappy customer is almost never pleased.

BUSINESS AND THE BEAST

Software is a strange beast. Your job as a salesperson doesn't stop when the contract is signed—it only begins. When you are done with the work, you deliver magnetic bits, fleeting images on phosphor, and words and diagrams on paper. Your customer must be convinced that these abstract items are worth their considerable cost.

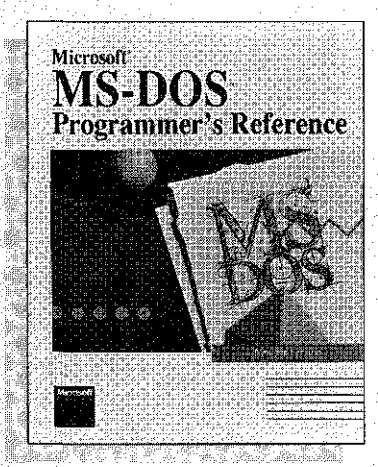
Throughout the project, you must involve the customer. Give the software a personal meaning for them. When it finally works and is accepted, the victory belongs not just to the developers but to everyone involved. Your job is to determine, early in the game, what the final system will be like, given the time and money available. You must mold the customer's expectations to match the product that will be delivered. If you have made the customer feel like a team member, almost any other problem with the project can be worked out.

Let me describe a project that was brought to its knees by poor management of customer relations. A large software development organization was assigned to develop a software system to test an advanced military system. The

William H. Roetzheim is a senior associate with Booz, Allen & Hamilton, and a noted author and speaker on software engineering topics. This column is based on his book, Structured Computer Project Management *(Prentice-Hall, 1988). Bill can be contacted at Booz, Allen & Hamilton, Inc., Suite 109, 4025 Hancock Street, San Diego, CA 92110; (619) 223-5681.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS Documented



Updated to MS-DOS version 5!

The Microsoft guide to MS-DOS® internals is back. This exclusive guide is now updated to include all versions of DOS from 3.3 through 5.0. This is the DOS book that gets used! Now available wherever computer books are sold. Or order directly from Microsoft Press.

$24.95

To place your credit card order, call

1-800-MSPRESS.

8am to 5pm Central Time.
Refer to ad AVG.

Microsoft PRESS®

The Authorized Editions

Microsoft Press, One Microsoft Way, Redmond, WA 98052-6399.
$32.95 in Canada.
Call Macmillan Canada. 416-293-8141.

project involved approximately $80 million worth of software, funded mostly by a joint services agency on the other side of the country. The program manager allowed a conflict-oriented relationship to develop between the project team and the customer team. The relationship deteriorated so much that the project team refused to communicate with the customer except at formal reviews (held about once a year). Instead of working with the customer's people and getting them involved in as many decisions as possible, the project manager refused to allow them even to review preliminary documents. What was the result? As you might expect, the customer used the formal reviews as an opportunity to obtain "proof" that the project team did not know what they were doing. After two years and approximately $20 million, the customer was upset enough to cancel the project outright—an almost total loss for all involved.

KEEP THE CUSTOMER POSTED

As a project manager, I once accepted a project to write a software program that would automatically monitor military tactical data links, parse the received information, and store the information in a database optimized for some tactical decision-making aids. Throughout the project, I handled critical design decisions by calling the customer, explaining the options, making a recommendation, and then following the customer's desires. Problems were explained carefully, and possible solutions were discussed. The customer was apprised of any risks as we became aware of them.

As the work progressed, it became apparent that the program would be late and over budget. We discussed this with the customer and were able to modify the requirements to allow delivery within the time and budget allowed. The project was a success both financially and in terms of customer satisfaction, all because the customer was on our team.

Once the decision has been made to use project management, it's important that every aspect of the project be given its due. Too many consultants fail to realize that careful management of customer relations can be as important as technical expertise.

Continued from Baker, page 36

it a *supervisor)* that translates the Windows 16-bit API into a 32-bit clone of Windows API for your 386 programs. Some Windows calls may overload a passed parameter as either a **long** or a **far** pointer. The supervisor layer must translate a far pointer segment:offset (32-bits) from Windows to a far pointer (48-bits) for the 386. Parameters need to be passed from this supervisor layer that Windows sees to the 386 segments that your application runs in. The challenge is making all this as transparent as possible for the developer. I can't say much in detail about these products until they're released (and I'm released from nondisclosure).

A separate DOS extender is not needed, because it is provided by Windows 3 and its DPMI services. Both MetaWare and Watcom have written a Windows virtual device driver so that their symbolic debuggers will run under Windows.

Watcom will release C/386 for Windows and FORTRAN 77/386 for Windows (call the company for current prices). MetaWare calls its product the Windows Application Developer Kit ($495), which must be purchased in addition to MetaWare's High C/386. Look for them soon.

HOT TIMES AHEAD

Theoretically, Windows should have supported being either a DPMI host or a client. However, if Windows were to run as a DPMI client, you could run all your favorite Windows apps under an OS/2 2.0 DPMI host. This would create some potential political and economic problems for Microsoft and Windows. So IBM has been left the formidable task of implementing the Windows API under OS/2 2.0.

These are exciting times for 386/486 development. Half the PCs sold today have either a 386 or 486 microprocessor. Hardware prices continue to decline. The prices of math chips (from Intel and Cyrix) have dropped dramatically. The DOS-extender market appears healthy, with many promising new products for the developer. Stay tuned for more ways to enhance this popular operating system.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A SOFTWARE FABLE

By Richard Riehle

Once upon a time two programming teams were hired to create a new piece of software in two languages. The Hare Team selected C, and the Tortoise Team chose Ada.

The leader of the Hares said, "We can win the race, because our language is lean and easy to learn. It isn't encumbered with Ada's rules and features. The Tortoises can never catch us, because they carry that big Ada shell around on their backs."

The Tortoise team felt sad. They knew the Hares could outrun them, and it seemed hopeless even to enter the race. The Hares were half finished before the Tortoises completed the package specifications.

It was a sight to see: code poured into source files like a tidal wave into a thimble. The Hares couldn't contain their excitement, and their zeal resulted in many wonderful, unexpected new system features. Compilation was lightning fast. The Hares' system could be recompiled twice as fast as the Tortoises'.

Everyone could see that the Tortoise team didn't have a chance. And, sure enough, the Hares won the race—by three months.

Then a funny thing happened: it turned out that the race was not over after all. A year after the system was accepted, the client decided to add some new features. But many of the Tortoises and Hares had gone on to other projects, so much of the original code had to be reread and understood anew. Also, some algorithms had to be completely rewritten.

The new team of Hares picked up the C source code and began to run with it. They ran very fast indeed. But the faster they ran, the less they understood, and they found the forest of C code very thick and dark. Soon, they were lost in the forest.

The Tortoises also formed a new team. They discovered that they could easily understand the well-encapsulated packages and separate compilations. The changes were made, tested, and implemented quickly. Then the Tortoises went on to do other projects.

Meanwhile, the more the Hares tried to fix the C code, the more the forest resembled a jungle. After a very long time, the lead Hare said, "This is no use. It would be better to rewrite the whole system from scratch." A year later, the leader of the Hares took another job, the other Hares were assigned to more pressing projects, and the revisions were deemed by management as unnecessary at the costs they were incurring.

So ends the tale of the Tortoises and the Hares.

Richard Riehle is a software engineer with AdaWorks. He can be reached at 1090-105 Tanland Drive, Palo Alto, CA 94303; (415) 858-1551. *This fable may be distributed freely as long as Richard Riehle and* Programmer's Journal *are credited.*

INSTALIT™

The Premier Worldwide Installation System℠

for DOS • OS/2 • Microsoft® Windows

INSTALIT™ has all the features you've requested over the last five years, so you won't constantly run into walls. It's capable enough to satisfy the world's most complex installation requirements, but so logical it's a joy to use from the start. Take 90 days to decide we're right or your money back. Use it with all your products with no royalties. Demo available!

• All versions share common readable, flexible, procedural, collapsible, encryptable, script language • Rugged, automatic, transparent error handling • Extensive system, environment and network sensing • Unsurpassed compression performance • Suppressible copyright message • Fully automated disk building with transparent huge file splitting • Sophisticated file queueing system. • Easy, flexible text/binary file modification • Easy pop-up windows/dialog boxes • And much, much more!!

"INSTALIT™ is the finest, most complete installation program . . . made the difficult seem trivial. What a delight. I have easily spent hundreds of hours maintaining, patching, kludging and tweaking our former installation program into coping with yet another program or database variation. No more! I'm FREE!"
— Bill Elliott, Context Software Systems, Burr Ridge, IL

"Excellent product. Easy to write very complex installs in a short period of time. . . .encourages you to try different things."
— Randy Jackson, CIS Technologies, Inc., Tulsa, OK

For DOS - $149
Specify national language
For Windows - $199
With C Source add $100
Multilingual for DOS $249
Speaks 10+ languages
VISA/MC/AMEX/PO/COD accepted
Shipping $5, next day $15

America's Favorite Installer℠
Orders only 1-800-448-4154

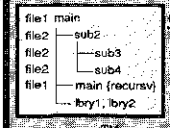
HPI

A Product of Helpful Programs, Inc.
P.O. Box 16078 • Huntsville, AL • 35802 • (205) 880-8782 Voice/FAX

Circle 8 on Reader Service Card

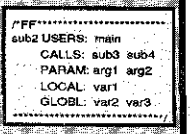
C AND C++ DOCUMENTATION

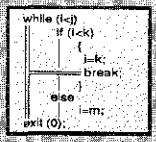
NEW! C-METRIC™ ($49) COMPLEXITY / QUALITY
• Calculates "Cyclomatic" path complexity for functions and system
• Counts Lines, Comments, Code, and 'C' Statements



C-CALL™ ($59) FUNCTION HIERARCHY
• Tree-Diagram Showing function hierarchy
• Table Of Contents of functions versus files
• Detailed cross-reference of functions

C-CMT™ ($59) FUNCTION COMMENT
• Generate/Insert function comment blocks
• Re-run anytime to update comment blocks
• Retains any user-generated comments





C-LIST™ ($49) LIST OR REFORMATS
• Action-Diagrams show logic/control flow
• Reformats to various standardized formats
• Line numbers, page numbers, and titles

C-REF™ ($49) CROSS-REFERENCES IDENTIFIERS
• Summary list of Local/Parameter/Global/Define Identifiers used
• Detailed Cross-Reference of Identifiers vs Function/Line usage
• Produces Class Hierarchy Tree-Diagram for C++ Classes

SPECIAL OFFER- ($189) Complete C-DOC™ Package
• All 5 programs fully integrated as 1 overall C-DOC program (DOS)
• NEW! special offer includes OS/2 protected-mode program
• Unconditional 30-DAY Money-Back Guarantee of Satisfaction !!

ACCURATE, and Fully AUTOMATIC (no source changes needed)

SOFTWARE BLACKSMITHS INC
6064 St Ives Way, Mississauga
ONT Canada L5N-4M1

VISA MasterCard
(416) 858-4466
Call for Immediate Shipment

Circle 9 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRESSROOM

PLUSFORT FORTRAN TOOLKIT

Polyhedron Software has introduced plusFORT, a Fortran toolkit that includes tools for restructuring, reformatting, static analysis, project management, and documentation. The restructurer, based on the company's SPAG program, can be used to rewrite spaghetti logic in structured style. Declaration standardization and variable renaming are built in. License fees start at $1290.

Distributed by OTG Systems, Inc., Route 106 & 374, Clifford, PA 18413, (717) 222-9100 (for Polyhedron Software Ltd., Witney, Oxon, UK).

Circle **166** on Reader Service Card

QUICKPAK AND PDQCOMM

Two new products from **Crescent Software** are designed for use with Microsoft Visual Basic. QuickPak Professional for Windows provides high- and low-level facilities, communications support, windowed terminal emulations, and tutorial information. Facilities include custom controls, predefined dialog boxes, data entry routines, and financial and scientific functions. Low-level routines include array manipulation, file operations, and keyboard services. PDQComm for Windows provides routines that access multiple communication ports from within a Visual Basic application. The low-level routines in both products are written in assembly language. QuickPak Professional is $199; PDQComm is $99.

Crescent Software, Inc., 32 Seventy Acres, West Redding, CT 06896, (203) 431-4626.

Circle **167** on Reader Service Card

MS-DOS 5 AND PROFILER

Microsoft has released MS-DOS 5. With 80286 or higher PCs, MS-DOS 5 can reside in the HMA, providing 45K of additional memory for DOS applications and as much as 621K of free conventional memory. Users can load TSRs, device drivers, and network drivers into upper-memory blocks on 80386 and 80486 machines, freeing approximately 133K of memory. MS-DOS 5 provides a new MS-DOS Shell menu interface, which includes a file manager, program manager, task swapper, utilities, and on-line help. New utilities include undelete (retrieves deleted files) and unformat (reverses formatting on newly reformatted disks). A full-screen editor has pull-down menus and supports a mouse. Upgrading requires the presence of DOS 2.11 or later versions. MS-DOS 5 Upgrade is $99.95.

Microsoft has also released Source Profiler. This profiler supports DOS, Windows, and OS/2 platforms. It helps developers test and optimize applications written using Microsoft language products by generating data about which lines and functions are executed most often. The Source Profiler uses 80K of memory and costs $79, or $50 for registered users of Microsoft languages products.

Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399, (800) 992-3675 or (800) 426-9400.

Circle **168** on Reader Service Card

F77L VERSION 5.0

F77L Version 5.0, from **Lahey Computer Systems,** is a DOS 640K Fortran language system. Version 5.0 includes multilingual compiler and runtime error messages. New statements in this Fortran 90 product allow users to create smaller executables by dynamic allocation of arrays, write easily understood code, and improve program structure. New porting features support moving mainframe code to and from PCs. The system includes an editor, a debugger, a profiler, the SLR linker, a library manager, Opus Make, and video graphics libraries. F77L Version 5.0 is $595; upgrades start at $150.

Lahey Computer Systems, P.O. Box 6091, Incline Village, NV 89450, (800) 548-4778 or (702) 831-2500.

Circle **169** on Reader Service Card

PAGED MEMORY MANAGER

The Paged Memory Manager (PMM) from **Emerging Technology Consultants** is a heap management system to improve the performance of Windows and OS/2 PM applications. PMM saves time and reduces the overhead incurred by function calls to allocate or lock memory. It allocates, locks, and unlocks fixed-size "pages" transparently and manages the sub-allocation of memory within those pages. Features include multiple and heap-level locking, memory grouping, and debugging and statistics functions. PMM sells for $295 for the OS/2 or Windows version, or $495 for both. Source code licenses are available for $2,500.

Emerging Technology Consultants, Inc., 3405 Penrose Place, Boulder, CO 80301, (303) 447-9495.

Circle **170** on Reader Service Card

OPTLINK/COMPRESS 3.0

The OPTLINK/Compress 3.0 linker, from **SLR Systems,** provides compressed, single-level overlays. It produces self-extract-to-run EXE files that are up to 50% smaller than /EXEPACK. The embedded decompression routine runs automatically at file execution. OPTLINK/Compress 3.0 can compress more than 8,000 overlays. It allows indirect calls, EMS/XMS caching, overlays to a separate file, calls from overlay to overlay, large- and small-model overlays, and automatic allocation of libraries. OPTLINK/Compress 3.0 is $350; upgrades are $75.

SLR Systems, Inc., 1622 North Main Street, Butler, PA 16001, (412) 282-0864.

Circle **171** on Reader Service Card

NDPWIN AND NDPWIN-SDK

MicroWay's NDPWIN Windows 3 extender provides 32-bit support for the Windows 3 platform on the company's protected-mode 386 compilers for Fortran, C, C++, and Pascal. NDPWIN implements character- and graphics-based I/O, allowing users to create or run a 32-bit character-based application in a window or create an application that calls the Windows API. NDPWIN supports DOS and Unix features that are not available in Windows 3. The NDPWIN-SDK provides 32-bit access to the Windows GUI. The NDP compilers are priced at $895 each; NDPWIN-SDK is $1495.

MicroWay, P.O. Box 79, Kingston, MA 02364, (508) 746-7341.

Circle **172** on Reader Service Card

PERSONAL MEASURE V. 1.55

Spirit of Performance, Inc., has released Personal Measure Version 1.55. Personal Measure, which is compatible with MS-DOS 5, monitors the performance of PC hardware and software, pinpoints processing bottlenecks, and helps users tune overall system performance. The product, which is compatible with

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

real, standard, and enhanced modes of Windows 3, also shows how network resources are used by programs. The TSR program that collects hardware and software performance data occupies less than 9K of memory. Personal Measure 1.55 is $125; upgrades are $50 for releases before Version 1.5, or $25 for later releases.

Spirit of Performance, Inc., 73 Westcott Road, Harvard, MA 01451, (508) 456-3889.

Circle 173 on Reader Service Card

QNX 4.0

Quantum Software has released QNX 4.0, an IEEE POSIX-compliant OS based on microkernel technology. The microkernel handles server tasks such as file handling, device I/O, and network management separately from the kernel, which executes scheduling tasks and coordinates message passing. QNX is a Unix-like OS for Intel-based PCs. QNX 4.0 is $795.

Quantum Software Systems Ltd., 175 Terrence Matthews Crescent, Kanata, Ontario, Canada K2M 1W8, (613) 591-0931.

Circle 174 on Reader Service Card

SMALLTALK/V WINDOWS 1.1

Digitalk's Smalltalk/V Windows Release 1.1 combines the company's object-oriented programming environment with Microsoft Windows 3. Smalltalk/V Windows includes source code browsers, inspectors, and push-button debuggers, as well as interfaces to DDE and DLLs. No runtime royalties are charged. Smalltalk/V Windows Release 1.1 is $499.95; upgrades are $25.

Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045, (213) 645-1082.

Circle 175 on Reader Service Card

TOPAZ VERSION 3.0

Software Science has released the source code to its Topaz version 3.0 database management software. Topaz allows users to write relational database applications with standard dBASE file and syntax using Turbo Pascal or QuickPascal. The product supports full-screen editing, dBASE DBF files, indexing, report generation, a memo and text editor, search capabilities, print spooling, and pick and tag lists. With an executed license agreement, source code for Topaz Version 3.0 is $499. Registered Topaze owners can buy the source for $399.

Software Science Inc., 100 Valley Drive, Brisbane, CA 94005-1350, (415) 467-7077.

Circle 176 on Reader Service Card

WIN++

Win++, by **Blaise Computing Inc.**, is a Windows class library designed specifically for use with Borland C++. Win++'s extensive class libraries provide a foundation for Windows applications. Most of the classes provide a simplified, high-level C++ interface to Windows, and others can be used for representing internal data structures and integrating data into applications. Win++ provides a class for all common types of display objects in Windows that are accessed by handles, and it supports the construction of DLLs. Win++ costs $249, including source code.

Blaise Computing Inc., 819 Bancroft Way, Berkeley, CA 94710, (415) 540-5441.

Circle 177 on Reader Service Card

ASYNC PROFESSIONAL

TurboPower Software has released Async Professional, an object-oriented tool library to ease development of asynchronous communications applications in Turbo Pascal. Using a layered architecture, Async Professional provides both object-oriented and traditional procedural interfaces. All standard file transfer protocols are supported. Async also features hardware flexibility and debugging tools. Async Professional is $139.

TurboPower Software, P.O. Box 49009, Colorado Springs, CO 80949-9009, (800) 333-4160.

Circle 178 on Reader Service Card

GUI_MASTER

GUI_Master from **Vleermuis Software Research** is a program constructor for C++. This object-oriented toolkit allows faster development of event-driven, window-based programs in the OS/2 PM and MS Windows environments. Both versions are available for the Zortech and Glockenspeil C++ compilers; the MS Windows version for Borland's C++ compiler is in beta-test. GUI_Master has three parts: the Interface Builder, the Browser, and the Class Tree (which has 85 GUI interface classes). GUI_Master OS/2 PM costs $495, and the price for the MS Windows version is $545.

Vleermuis Software Research, P.O. Box 2584, 3500 GN Utrect, The Netherlands, 31/30 31 04 26 (fax).

Circle 179 on Reader Service Card

WATCOM C/386 FOR WINDOWS

Watcom's new C/386 for Windows enables the development of 32-bit Windows 3 GUI applications. It has an interactive debugger and is ideal for porting 32-bit Unix and Mac applications to Windows 3. C/386 is compatible with Microsoft library and source. It is selling for $2,250, or $1,295 for upgrades that allow 32-bit applications in DOS or Windows.

Watcom, 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2, (800) 265-4555.

Circle 180 on Reader Service Card

VISUAL COMPARE

Visual Compare from **Whitney Software, Inc.**, lets you see the differences between two program source files in a full-screen display. Differences between files can be selectively discarded. Visual Compare's scrollable display highlights common, deleted, and inserted lines. The price is $30 for DOS or OS/2, or $45 for DOS and OS/2.

Whitney Software, Inc., P.O. Box 4999, Walnut Creek, CA 94596, (415) 933-9019.

Circle 181 on Reader Service Card

C-DEBUG

Softran Corporation is offering C-Debug, a preprocessor for C language programs. C-Debug works with any C compiler and hardware or operating system. It alerts users to data corruptions in memory areas. Errors are located and described, minimizing guesswork. C-Debug for DOS is $249, for Novell $349; prices for other operating systems are available from Softran.

Softran Corporation, One Naperville Plaza, Naperville, IL 60563, (800) 462-3932.

Circle 182 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY II** "The Lock"

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
201 East Franklin, Suite 11
Richmond, MO 64085

(800) 227-0644 **(816) 776-2700**
FAX (816) 776-8398

Circle **110** on Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle **111** on Reader Service Card

Duplicating disks? Read this:
If not, please forward to someone who is.

EZ-DISKCOPY PRO
Duplicates Disks Fast!

- Read source disk only once.
- Quickly & flawlessly duplicates 360K, 720K, 1.2M, 1.44M disks on your own PC/XT/AT/etc. ©
- Up to 300 disks/hr/machine!
- Formats, copies, verifies, stores images (optionally "crushed"), etc.
- Pre-formatted are even faster.
- More Speed? Just Lic. machines.
- Serialize and print labels too!
- Replaces $1000s in hardware.

ONLY $149+S&L (CP•1) Multi-lic. avail.

EZX P.O. Box 58177, Webster, TX 77598

Discvr/VISA/MC/AX Orders @1.800.800.2468
INFO: 713.280•9900 FAX•0525
TestDrive with Full Manual is only $20 +S&I

Circle **112** on Reader Service Card

New Version 2 - Execute up to 256 C Functions Concurrently In One Program with Conquerrent C

- **Great for real-time, data acquisition, clock displays, background printing, simulation, process control, games.**
- **Transparent reentrancy protection for most large model library functions.**
- **Flexible stack location and size management for each task.**
- **All memory models supported in Turbo and Microsoft C on DOS 80x86 PC's.**
- **New Low Price - $45 complete, no royalties, 30 day satisfaction guarantee.**

Available from:
Interstitial Software Systems
321 E. Buckingham Rd., Ste. 140
Garland, Texas 75040

Circle **113** on Reader Service Card

QUALITY TOOLS

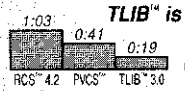
Memory Overlay Manager - transform your C, Clipper or Clarion application into a 5K TSR with hot-keys.....$99.95
Memory Master - break 640K barrier, one heap or many, EMS/XMS.....$59.95
ASMplus - structured ASM lang. & lib. Code ASM just like C or Pascal....$99.95
"C"erious Toolkit Plus - over 250 highly optimized fcts, C & ASM source...$99.95
386PM protected mode ver.....$199.95
Phase One - 100% replaceable plug-&-play C engine for Paradox, runs 4-10X faster, DOS and DDL ver. incl.....$495.00
PALCom - PAL lang. compiler. Create stand alone .EXE programs.....$1495.00

Please call our BBS at 516-331-6682 to try out our FREE demo programs.

TSR Systems Limited
116 Oakland Avenue
Port Jefferson, NY 11777
Ph: 516-331-6336 FAX: 516-331-6377

Circle **114** on Reader Service Card

TLIB™ is FASTEST!



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with **Opus' MAKE & Slick' MAKE.**

MS-DOS $139, OS/2 $195 + shipping. Visa/MC. 5 station LAN license $419 (OS/2 $595), call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 (919) 233-8128

Circle **115** on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle **116** on Reader Service Card

BGI Printer Drivers

The BGI Printer Driver Toolkit provides BGI drivers for Epson 9 & 24 pin, HP LaserJet II, and HP PaintJet printers. Use the BGI graphics interface to effortlessly generate high resolution hardcopy with Turbo C, TC++, and Turbo Pascal. **Full driver source code included.** $89.95

PC Timer Tools

60 functions implement microsecond resolution timing, precision delays, interrupt profiling, and timer tick management. Ideal for profiling and data acquisition/process control. Supports TC, TC++, MSC, TP. **Full library source code included.** $49.95

Prices postpaid USA, elsewhere add $4
VISA & MasterCard accepted.
30 day money back guarantee

RYLE DESIGN
Purveyors of Big Science
PO Box 22, Mt. Pleasant MI 48804
(517) 773-0587 CIS:73047,1765

Circle **117** on Reader Service Card

FORTRAN

FORTRAN Toolkits

Comprehensive libraries of over 300 graphics, text, and systems service subroutines from ellipses to child-processes to I/O verification. High-speed video drivers, from CGA to Super VGA, are all internal. Option rich, easy to use. No royalties. 300+ pg. manual, 8,000 lines of sample source. Price $215.

DOCUMENT'er and DIAGRAM'er

Makes source code understandable. Indexes, cross-references, and diagrams the structure of source code. No changes to source required. Nested includes, recursion, and annotations handled automatically. Price $165.

IMPULSE *Engineering*
PO Box 190206,
San Francisco, CA 94119-0206
(415) 788-4611

Circle **118** on Reader Service Card

THE CD-ROM DEVELOPER'S LAB™

Comprehensive CD-ROM multimedia production resource for PC and Apple developers. Proven design, management, data prep, programming, premastering and manufacturing techniques. Important specs from leading companies. Demos of off-the-shelf tools for imaging, audio and animation. Sample applications demonstrate the power of our source tools in C, Turbo Pascal. PC or MAC @ $395.00; Transportable version @ $425.00, Visa/MC accepted.

SOFTWARE MART, INC.
3933 Spicewood Springs Rd., Suite E-100
Austin, Texas 78759
(512) 346-7887 FAX: (512) 346-1393

Circle **119** on Reader Service Card

C/ANALYST

Can quickly bring you up to speed on unfamiliar C source code.

Call (503) 581-5622
For Free Demo

CATER SOFT WARE

2182 Westfarthing Way NW
Salem, OR 97304

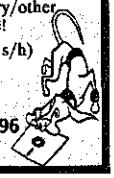
Circle **120** on Reader Service Card

INTEL® 80486 MACRO DISASSEMBLER

MD86 (Masterful Disassembler) is the most comprehensive macro disassembler on the market. Interactive by design not after-thought. Features include:

- **User defined macros with arguments**
- **Auto code/data separation**
- **Auto commenting and label naming**
- **Command menus and help screens**
- **Customizable disassembly options**
- **Comprehensive, indexed, manuals**
- **Fully supports 8086/87 thru 80486/87**
- **COM/EXE/SYS/memory/other**
- **Cross reference and more!**

Still only $67.50 + tax ($1.50 s/h)



C.C. SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Contact this advertiser directly.

Get the attention your product deserves.

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Display ads are 1⅝" wide by 2⅜" high. For more information or to reserve a space, call our ad department at **(800) 234-0386.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

LOOK

at the savings in development time you can get with DOCZ to automate subroutine and program documentation. Discover how to use DOCZ to build an object-oriented development environment to share and re-use your software modules. FREE MSDOS demo disk and samples. 60-day money-back guarantee. VISA/MC accepted. $195/ Single-User. Start benefiting NOW!

DOCZ™

Do It Better With Toolz!

Software Toolz, Inc.
8030 Pooles Mill Drive
Ball Ground, GA 30107-9610
(404) 889-8264
Voice-mail: (800) 869-3878

Circle 121 on Reader Service Card

UPPER DECK EDITOR FOR WINDOWS 3.0

Regular expressions, search across multiple files, full multi-level UNDO, Windows clipboard support, keyboard macros, compile within editor, customizable. Introductory price $75 plus $5 S/H (outside USA $15). CA residents please add sales tax. 30-day money-back guarantee.

Upper Deck Systems
P.O. Box 2751
Escondido, CA 92033
(619) 741-1075

Circle 122 on Reader Service Card

_miniEd — Add Text Editing to Your 'C' Applications (BASIC and PASCAL too!)

Ultra-small (less than 5K), full-featured window text editor designed to work with 'C', PASCAL or BASIC programs. _miniEd can be linked with your programs, or can be configured as a TSR and accessed via a software interrupt. Ideal for obtaining text input when RAM considerations are important or use VIEW mode as the display driver for your HELP system. All text and formatting information is held in a structure that is shared between your program and _miniEd.

+Written in 8086 assembler
+Uses any program-defined window size
+No Royalties
+Non-destructive VIEW mode
+Re-entrant/Edit multiple windows

_miniEd Object, Commented Source and Documentation: $99.00

Phone: 708-778-6060
Fax: 708-778-6063

Strategic Software Designs, Inc.
6 S 235 Steeple Run Drive – Suite 12B
Naperville, IL 60540

Circle 123 on Reader Service Card

If Your App Never Swaps, Stop Reading This Ad

However, if you are beset by other apps stealing **your** memory, or by real mode users, then your app occasionally crawls because of disk thrashing. There simply isn't enough memory to run comfortably, even with virtual memory and object caching.

If this sounds familiar then you need ***SegMentor***™. It optimizes your segmentation map so your app screams even when your user's memory is scarce.

SegMentor™ traces the entry point, exit point, and timing data for absolutely every one of your functions in a typical runtime session(s).* Then, it employes powerful algorithms, like those for auto-routing printed circuit boards, to explore the solution space for an optimal segmentation map. You get a more compact app with the fewest inter-segment (disk thrashing) calls. We even put the pragma directives for this new map in an include file for easy linking back into your code.

See feature article in March 1991 Microsoft Systems Journal (pg 49)

*If this profiling looks illuminating compared to your statistical analyzers and source code filters, call for more information on ***Performance Tracer***™. It gives you a **precise** and **complete** view of **exactly** what occurs in your app at runtime — call trees, object sizes, call counts, function times, etc.

MicroQuill INC. ***Performance Tuning Tools***
4900 25th Ave. NE 206 • Seattle, WA 98105
800 441 7822 • 206-525-8218 FAX 206-525-8309

Circle 124 on Reader Service Card

DEBUG WITHOUT SOURCE

Quaid Analyzer debugs by showing interrupts as they occur. You can even watch DOS boot load without hardware changes. The manual tells how to deal with programs that don't want to be traced - we use it on copy protection. You scroll through memory like text in an editor, and never type a command. US $200.00

Quaid Software Ltd Dept PJ
45 Charles St East Third Floor
Toronto Ontario Canada M4Y 1S2
(416) 961-8243
FAX: (416) 961-6448

Circle 125 on Reader Service Card

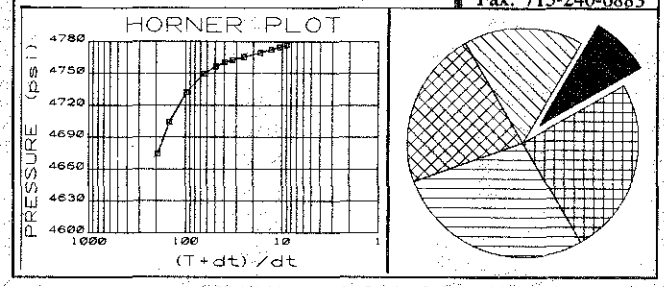
INGRAF 5.00

A Multi-Device Graphics Library for Scientific, Engineering and Business applications. Supports Video(CGA, EGA, VGA & Hercules), Printers (EPSON, Okidata, HP LaserJet, HP Paintjet & Postscript) and Pen Plotters(HPGL and DM/PL).

C, FORTRAN-77, PASCAL, MS Professional BASIC and QuickBASIC bindings.

FULL SOURCE CODE - NO ROYALTIES - $350.00

Sutrasoft
10506 Permian Dr.
P.O. Box 1733
Sugar Land
TX 77487-1733
U.S.A.
Tel: 713-491-2088
Fax: 713-240-6883



Circle 126 on Reader Service Card

PJ CODE IS AVAILABLE!

All the source code from the articles in this issue is available on disk from

Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
800-234-0386
503-747-0800

Code disks are $10.00 each. Please see page 81 for ordering information.

PJ Code Diskscriptions are available for $39.95. Look on the card opposite page 16 to order.

The code listings are also available for downloading from several Bulletin Board Systems around the country. See page 48 for listings.

Contact this advertiser directly.

Useable User Guides!

Your customers deserve user guides they can understand. For writing samples and a free estimate, call:

Extra Mile Writing Services
(619) 672-3388
11057 Carlota Street
San Diego, CA 92129

Circle 127 on Reader Service Card

FORTRAN TO STRUCTURED C / FORTRAN
TRANSLATOR

FORTRAN PROGRAMMER'S ASSISTANT II converts spaghetti FORTRAN into neat, maintainable C code. Run-time library source and C tool set included. It will make your conversion process easy and will significantly reduce your conversion and future maintenance costs. ASSISTANT II Plus also produces structured FORTRAN. Easy to use. At low prices. A must re-engineering tool for FORTRAN programmers.

MICROTOOLS
P.O. Box 2745 PJ
Santa Clara, CA 95055-2745
(408) 296-3478
FAX (408) 243-7688

Circle 128 on Reader Service Card

ATTENTION Software Writers!

Is your software innovative, usable and needed? Are your applications practical, thorough and well tested? Is your documentation well organized and understandable? InSync, a division of an established international software distributor, is evaluating new and innovative software to publish and market domestically and internationally. We are seeking new off-the-shelf MS/PC-DOS software applications (no accounting or inventory, please). ORIGINALITY IS A MUST for all submissions.

Please call our Evaluations Department to discuss submission procedures.

InSync software
1-800-783-7053

Circle 129 on Reader Service Card

Coming Next Issue:
OBJECT-ORIENTED PROGRAMMING

November/December 1991 Vol. 9.6
Advertising space reservations are due:
Sept. 6, 1991
Materials are due:
Sept. 13, 1991

Please call 503-747-0800

Ask for Jay or Maryann in the advertising department to reserve your space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Playing for keeps

By Hal W. Hardenbergh

Successful companies, like successful people, tend to gather enemies. It's no accident that Microsoft and Intel are currently being investigated for allegedly unfair business practices (one assumes that Compaq and Lotus are next in line).

Back around 1970, some officials of the U.S. Department of Justice decided that IBM was unfairly monopolizing the U.S. computer market. For 10 years, Justice tried to break up IBM. About the time Reagan was elected, somebody finally figured out that the United States needed a strong computer company in the international, not the national, market. And so the suit was quietly dropped.

IBM was only the first company Justice was trying to break up. The second company on their list was General Motors, and GM was running scared. Unlike IBM, whose sales were almost all based on the 360/370 architecture, GM would have been simple to break up: just break out each division (such as Chevrolet) as a company. In 1970, each GM division was making its own engines, transmissions, and so forth.

GM decided on a defensive reorganization. It spent billions and billions of dollars to remake the company. It created one engine division that made engines for all of GM's cars. It created one transmission division. The reorganization apparently fulfilled its objective, because Justice never did attempt to break up GM.

An unfortunate side effect of the reorganization was that all GM cars began to look pretty much alike, for the very good reason that they *were* alike. Why buy an expensive Cadillac when it's just a Chevy with different chrome trim? That, along with the 1974 oil crisis, accomplished Justice's goal of reducing GM's domination of the U.S. market, with no court proceeding necessary. It also started our "balance of trade" problem, because it gave people a reason besides fuel economy to buy a foreign, not a GM, car.

The point is that GM's reorganization was a protective move to defend itself from our legal system. Almost 20 years later, Intel is making similar protective moves, for the same reason. This is a wise decision on Intel's part; our legal system is even more arbitrary and capricious these days than it was when Justice made IBM spend billions of dollars to defend itself, only to abandon the antitrust lawsuit in the end.

ILLUSTRATION BY JIM CARPENTER

In November 1981 there was a wedding between Intel and AMD. IBM, which wanted a second source for the 8088 used in its PC, held the shotgun. This "wedding" was a contractual arrangement scheduled to run for 10 years; it will expire shortly after you read this. Like many weddings these days, this one didn't "take." The divorce proceedings are not just unseemly but downright vicious.

The contract called for AMD to get rights to Intel microprocessors in exchange for AMD products of equal technological (but not necessarily commercial) value. At the time the contract was signed, a microprocessor could be developed for about $100,000. These days, the cost is more than 100 million dollars. Intel did not believe that AMD upheld its end of the contract, so it withheld the 386 design from AMD. Obviously, AMD disagrees. If this contractual dispute were to be fought in regular civil court, the lawsuit would be in progress into the next century. Intel and AMD, to their credit, decided that arbitration was the sensible way to go. So they found a retired judge who was agreeable to both parties to serve as the arbitrator (aka rent-a-judge), and the three parties negotiated a contract. *Another* contract, that is.

This three-party contract specifically limited the amount of dollar damages that the arbitrator could award to either party. One of the first things the arbitrator did—I am not making this up—was to *file suit in civil court to increase by an order of magnitude the amount of monetary damages he was allowed to award!* How could he do this? This is a contract dispute; the three-way contract was between Intel and AMD on the one hand, and the arbitrator on the other!

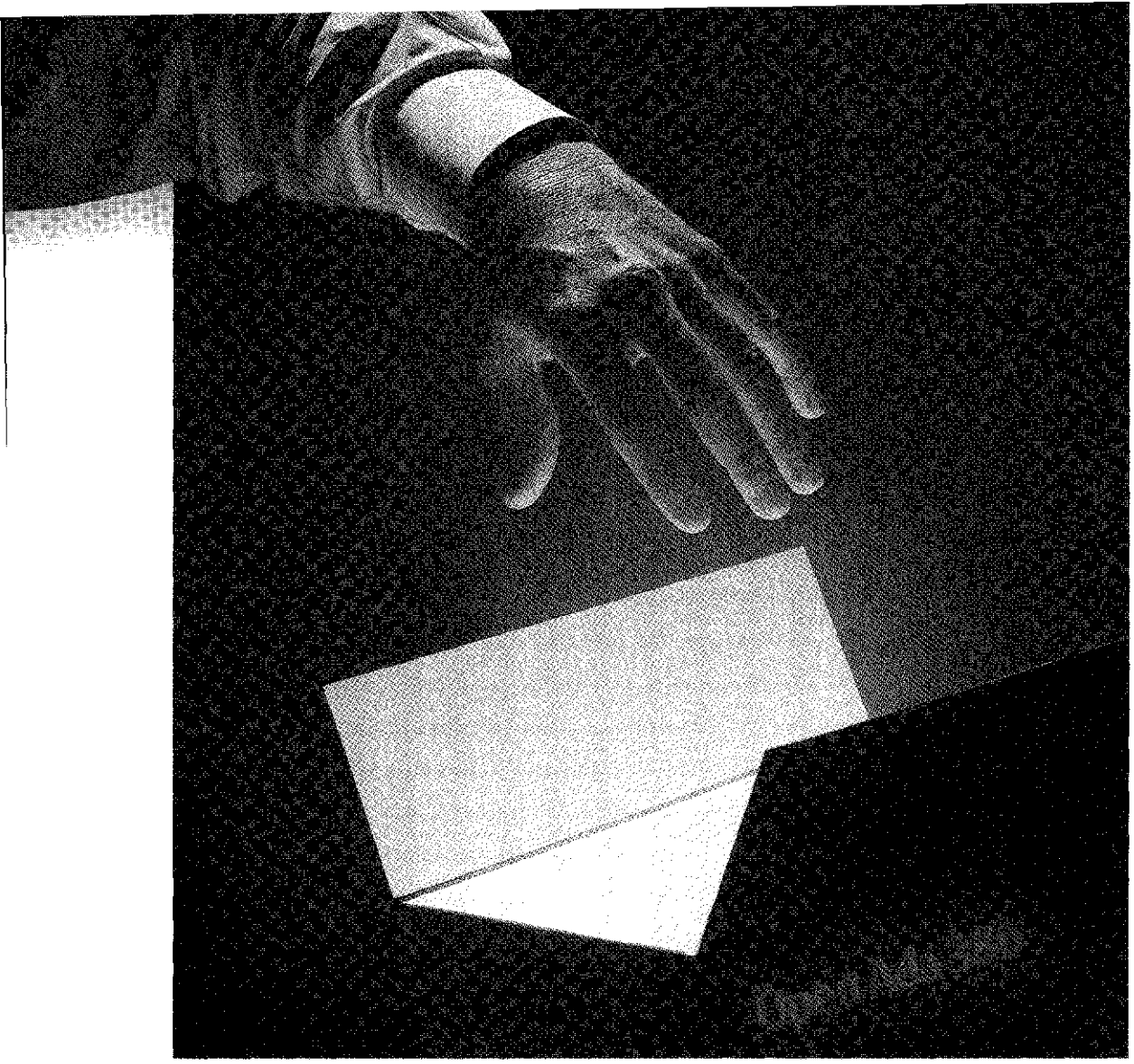
Yet the purpose of the arbitration was to stay out of civil court! Yikes! Are you beginning to see why some folks think our legal system is out of control? (An aside: most people think justice is what you're supposed to get in court. No, that's what you're supposed to get in *criminal* court. In civil court what you're supposed to get is law. If you're one of the complainants, you won't like the law. Ask any veteran of a messy divorce proceeding.)

The arbitrator has also refused to rule out the possibility of awarding Intel microprocessor designs to AMD, even without the contractually required even-technology exchange. *That's* why you're seeing so many different versions of the x86 architecture in Intel's product line: the supposition is that the arbitrator surely wouldn't award Intel's entire portfolio to AMD.

So the 50-MHz 486 is not just a shrunk 486. In addition to moving from 1 micron to 0.8 microns, the chip was actually redesigned, using three layers of metallization rather than two. I count five distinct products in Intel's x86 line now: the 386DX, the 386SX, the 386SL, the standard 486, and the new 50-MHz 486 (the 486SX and 487, for now, are pin-out variants of the standard 486).

All of us have an interest in the continuing development of the x86 product line. No company, not even Intel, can afford to spend hundreds of millions of dollars for future versions of the x86 architecture only to be forced to give its product to a competitor. AMD has lots of fans—perhaps including you?—in its corner. Be careful what you wish for; you might get it.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



OpenAda DOS386

At last, 386/486 Ada development power and performance within your reach

$495 INTRODUCTORY PRICE

Meridian's new OpenAda® DOS 386 proves that a top of the line development system doesn't have to threaten your bottom line. At $495 it challenges desktop and workstation products with true 386/486 power and performance, a 32 bit development environment, and the ability to target both 16 and 32 bit applications.

Economical—No other product responds so clearly to the federal government's push to secure high performance PC tools targeting 16 and 32 bit applications—within budget restrictions. Coupled with the June mandate requiring Ada for all Department of Defense projects, this capability makes OpenAda DOS 386 a necessary choice for serious programmers. The price makes it an easy one.

Proven—OpenAda DOS 386 represents a new generation of a proven product line that has enjoyed unprecedented popularity, selling over 10,000 copies within just 6 months. Products that *PC Week* said "slashed the price and raised the standard for PC based Ada development."

Friendly—Continuing the OpenAda tradition, OpenAda DOS 386 adds new tool support to a lengthy list of open and friendly advantages, including a pretty-printer and a cross-reference generator. In addition, it also provides extended support for both Microsoft® and the Metaware® C compilers.

Powerful—For all its ease of use, OpenAda DOS 386 takes full advantage of the resources of 386/486 architecture, providing you with the power and performance you need in a front end development system.

Complete—OpenAda DOS 386 includes a complete set of development tools, including:

■ Automatic recompilation tool ■ Ada source level debugger ■ on-line help ■ library management tools ■ complete documentation ■ graphic and enhanced numeric libraries ■ ACE User Interface

Limited Offer—To remain at the forefront of software engineering you have to explore Ada and Meridian has made it available for only $495. But you have to act now. This special offer lasts only 60 days. After that, you'll have to pay $1695 for this exciting product.

At any price, OpenAda DOS 386 is an investment in your future. Meridian's renowned customer support is always available. OpenAda products for PC DOS, UNIX and Macintosh Platforms start at $149.

Order now to OpenAda DOS 386. **Call 1(800) 221-2522.**

Meridian Software Systems, Inc., 10 Pasteur St., Irvine, CA 92718

OpenAda is a registered trademark of Meridian Software Systems, Inc.
All product names are registered trademarks of their respective manufacturers.

OpenAda DOS386

Circle **56** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Who would you expect to give you the best compiler for creating Windows applications?"

—Answer below

Introducing a vastly superior way to write applications for Microsoft® Windows. New Borland® C++. The only complete C and C++ programming environment for Windows *and* DOS. Borland C++ generates Windows applications and DLLs, supports Windows debugging, and includes a visual resource editor. So you don't need to buy the Microsoft Windows Software Development Kit (SDK).

Borland C++ is from the people who know what professional programmers want. In fact, Turbo C++ Professional (the predecessor to Borland C++) won both *PC Magazine*'s 1990 Technical Excellence Award and *BYTE*'s 1990 Award of Excellence.

Designed for professionals by professionals

Every copy of Borland C++ comes with:

- ANSI C compiler and C++ compiler
- Turbo Drive™ compiler and integrated development environment running in protected mode for huge capacity
- Precompiled headers that significantly increase recompilation speed
- Turbo Debugger® for Windows and DOS
- Whitewater™ Resource Toolkit for visually creating icons, dialogs, bitmaps and menu bars
- Turbo Profiler™ and Turbo Assembler®

Compare these costs

	Microsoft® C 6.0	**Borland C++**
C compiler	$ 495	$495
C++ compiler	not available	included
Windows programming tools	$ 495* (SDK)	included
Assembler	$ 150	included
Programming Windows	$ 30	included
Total	$1,170	**$495****

Buy now! Get *"Programming Windows"* free!

Buy Borland C++ now from your local dealer and get a free copy of *Programming Windows* by Charles Petzold, the best-selling book about learning Windows programming.†

Special upgrade pricing is available from Borland for owners of Turbo C,® Turbo C® Professional, Turbo C++ and Turbo C++ Professional.

See your dealer today, or call Borland to upgrade at 1-800-331-0877

Answer:

BORLAND

The Leader in Object-Oriented Programming for Windows and DOS

CODE: MC82

*Effective price is $254 if SDK is purchased bundled with Microsoft C. **$495 is the suggested retail price. †Offer good in U.S. and Canada while supplies last. Upgrade customers not eligible for free book. Copyright © 1991 Borland BI 1397A

Circle **57** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED